175 PTAS
mi computer
62
CURSO PRACTICO DEL ORDENADOR PERSONAL,
EL MICRO Y EL MINIORDENADOR
PRESS
HERO 1
MAPLIN
Editorial Delta, S.A.

MI COMPUTER CURSO PRACTICO DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

Publicado por Editorial Delta, S.A., Barcelona

Volumen VI-Fascículo 62

Director:	José Mas Godayol
Director editorial:	Gerardo Romero
Jefe de redacción:	Pablo Parra
Coordinación editorial:	Jaime Mardones
	Francisco Martín
Asesor técnico:	Ramón Cervelló

Redactores y colaboradores: G. Jefferson, R. Ford, F. Martín, S. Tarditti, A. Cuevas, F. Blasco

Para la edición inglesa: R. Pawson (editor), D. Tebbutt (consultant editor), C. Cooper (executive editor), D. Whelan (art editor), Bunch Partworks Ltd. (proyecto y realización)

Realización gráfica: Luis F. Balaguer

Redacción y administración:
Paseo de Gracia, 88, 5.°, 08008 Barcelona
Tels. (93) 215 10 32 / (93) 215 10 50 - Télex 97848 EDLTE

MI COMPUTER, *Curso práctico del ordenador personal, el micro y el miniordenador*, se publica en forma de 96 fascículos de aparición semanal, encuadernables en ocho volúmenes. Cada fascículo consta de 20 páginas interiores y sus correspondientes cubiertas. Con el fascículo que completa cada uno de los volúmenes, se ponen a la venta las tapas para su encuadernación.
El editor se reserva el derecho de modificar el precio de venta del fascículo en el transcurso de la obra, si las circunstancias del mercado así lo exigieran.

ISBN: 84-85822-83-8 (fascículo) 84-7598-034-7 (tomo 6)
84-85822-82-X (obra completa)
Depósito Legal: B. 52/1984

Fotocomposición: Tecfa, S.A., Pedro IV, 160, Barcelona-5
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona) 208503
Impreso en España - Printed in Spain - Febrero 1985

Editorial Delta, S.A., garantiza la publicación de todos los fascículos que componen esta obra.

Distribuye para España: Marco Ibérica, Distribución de Ediciones, S.A., Carretera de Irún, km 13,350. Variante de Fuencarral, 28034 Madrid.

Distribuye para Colombia: Distribuidoras Unidas, Ltda., Transversal 93, n.° 52-03, Bogotá D.E.

Distribuye para México: Distribuidora Intermex, S.A., Lucio blanco, n.° 435, Col. San Juan Tlihuaca, Azcapotzalco, 02400, México D.F.

Distribuye para Venezuela: Distribuidora Continental, S.A., Edificio Bloque Dearmas, final Avda. San Martín con final Avda. La Paz, Caracas 1010.

Pida a su proveedor habitual que le reserve un ejemplar de MI COMPUTER. Comprando su fascículo todas las semanas y en el mismo quiosco o librería, Vd. conseguirá un servicio más rápido, pues nos permite realizar la distribución a los puntos de venta con la mayor precisión.

Servicio de suscripciones y atrasados (sólo para España)

Las condiciones de suscripción a la obra completa (96 fascículos más las tapas, guardas y transferibles para la confección de los 8 volúmenes) son las siguientes:

a) Un pago único anticipado de 19 425 ptas. o bien 8 pagos trimestrales anticipados y consecutivos de 2 429 ptas. (sin gastos de envío).

b) Los pagos pueden hacerse efectivos mediante ingreso en la cuenta 6.850.277 de la Caja Postal de Ahorros y remitiendo a continuación el resguardo o su fotocopia a Editorial Delta, S.A. (Paseo de Gracia, 88, 5.°, 08008 Barcelona), o también con talón bancario remitido a la misma dirección.

c) Se realizará un envío cada 12 semanas, compuesto de 12 fascículos y las tapas para encuadernarlos.

Los fascículos atrasados pueden adquirirse en el quiosco o librería habitual. También pueden recibirse por correo, con incremento del coste de envío, haciendo llegar su importe a Editorial Delta, S.A., en la forma establecida en el apartado b).

Para cualquier aclaración, telefonear al (93) 215 75 21.

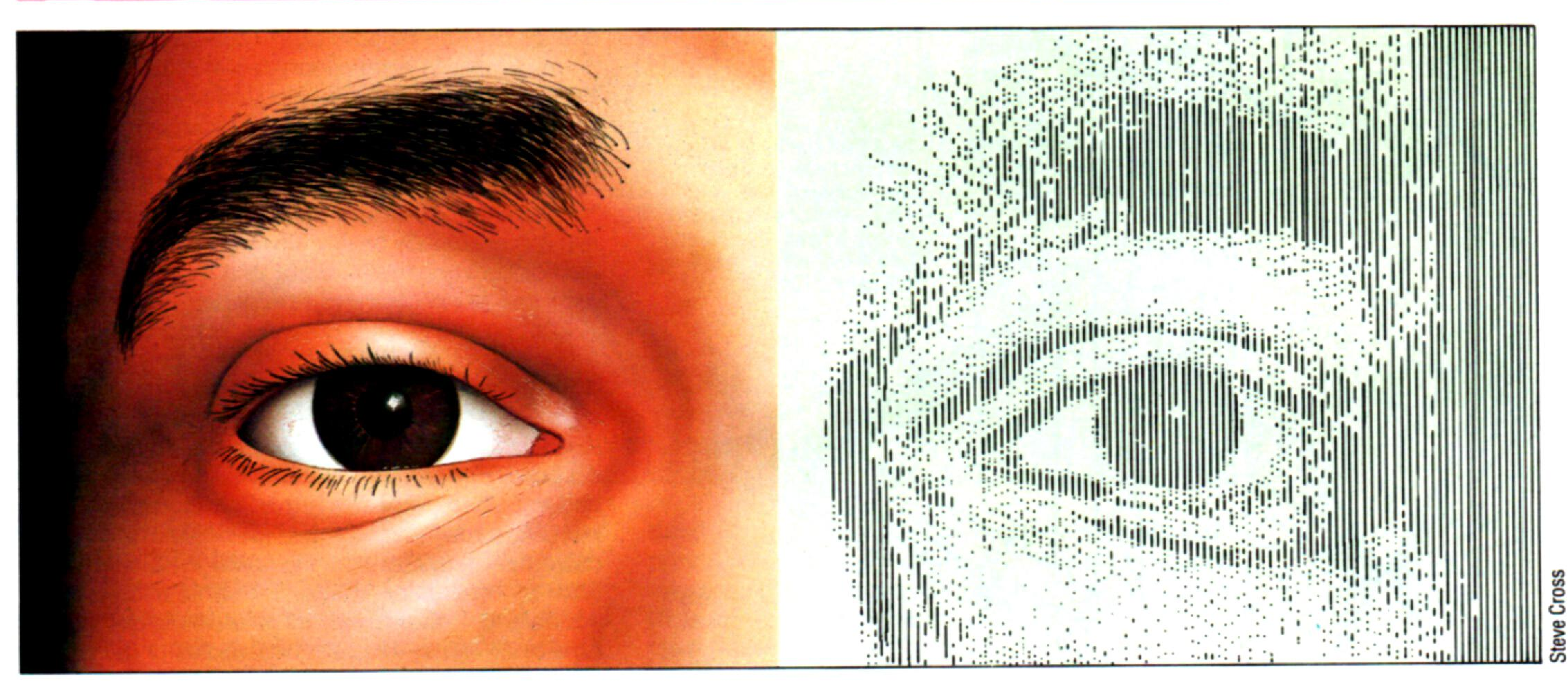
No se efectúan envíos contra reembolso.





Los ojos del robot

Examinemos los problemas que implica preparar un robot para «ver» los objetos del mundo exterior



La vista del robot y la vista del hombre
La naturaleza de la capacidad humana de ver se basa en gran medida en la interacción entre un complejo sistema de nervios y receptores, todos ellos procesados por el cerebro. Si bien una imagen visual se compone de patrones de luz y oscuridad marcados en la retina, el verdadero acto de "ver" tiene lugar en el cerebro. El cerebro de un robot procesa, asimismo, una imagen de patrones de luz y oscuridad, pero posee un grado de precisión muy inferior

De todos los sentidos, tal vez el más importante sea el de la vista. Las percepciones visuales son tan importantes para nuestra comprensión del mundo, que con frecuencia se utiliza la frase "es como tratar de describirle el color a un ciego" para ilustrar las dificultades que entraña explicar a alguien algo sobre lo que no tiene absolutamente ningún conocimiento. Sin la vista, nuestro conocimiento del mundo se ve seriamente restringido y, del mismo modo, un robot que no posea aparato visual está igualmente en situación de desventaja. Ya hemos visto cómo los robots utilizan sensores para detectar la presencia de un objeto en su camino; ahora deseamos desarrollar un sistema que los dote de un equipo visual tan eficaz como el del hombre.

El ser humano posee un iris que actúa a modo de lente, controlando la cantidad de luz que penetra en el interior del ojo, y una retina en la cual la lente focaliza la imagen. Pero el hecho es que el ojo en realidad no "ve" nada en absoluto; es sólo un transductor que convierte una señal en otra forma más aceptable. La verdadera tarea de ver la lleva a cabo el cerebro sobre la base de las señales que recibe desde sus sensores.

De modo que el tema de la vista del robot lo podemos dividir en dos partes claramente diferenciadas. La primera implica la construcción de un "ojo" adecuado que actúe como sensor para el sistema visual del robot; la segunda parte es el procesamiento de ordenador que se debe realizar antes de que el robot pueda darles algún significado a las señales provenientes de su sensor.

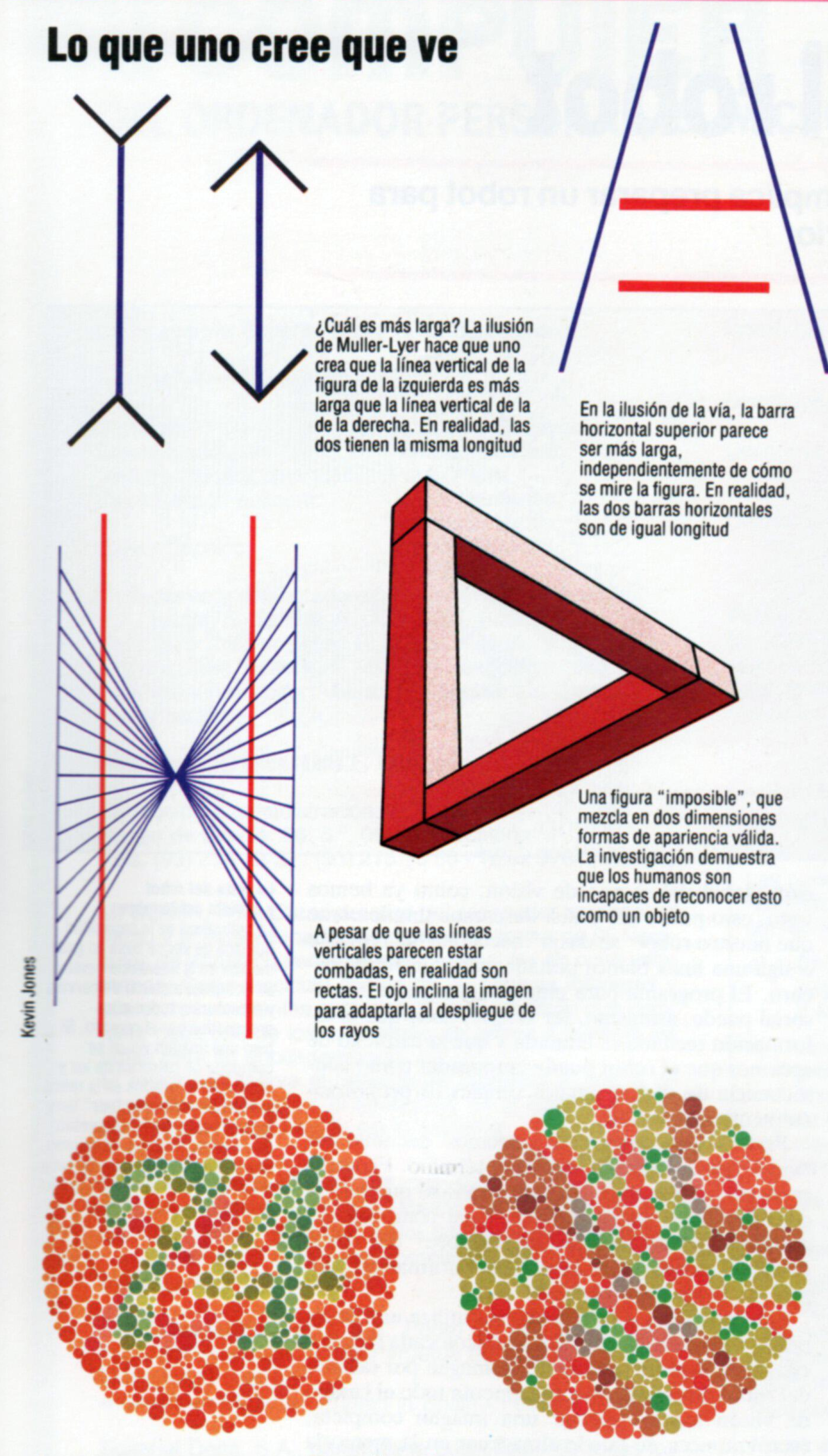
Construir un ojo robot no es demasiado difícil. En su nivel más simple, una célula fotoeléctrica puede actuar como una forma sencilla de ojo. Ésta puede dar una señal que corresponda a la iluminación global del campo de visión; como ya hemos visto, esto puede ser útil si deseamos simplemente que nuestro robot "se dirija" hacia una luz brillante o siga una línea blanca pintada sobre un fondo oscuro. El programa para emplear esta entrada sensorial puede, asimismo, ser simple, dado que la información recibida es limitada y que la cantidad de acciones que el robot puede emprender como consecuencia de dichas señales simples es proporcionalmente restringida.

Pero a esto apenas si podemos denominarlo "vista", en la acepción literal del término. Específicamente, necesitamos un sistema visual que pueda construir una imagen bidimensional completa del mundo, permitiendo que el "cerebro" del robot examine exactamente la misma información que procesa el cerebro humano.

Una respuesta a este problema utiliza una única célula fotoeléctrica con una lente colocada frente a ella. Ésta explora la zona de la imagen por delante del robot, barriendo mecánicamente todo el campo de visión hasta construir una imagen completa; ésta, entonces, se puede almacenar en la memoria del ordenador. En la práctica, lamentablemente, este método es lento y poco fiable.

En la mayoría de los casos, sin embargo, el ojo robot se compone de algún tipo de cámara de video. Esta cámara puede ser del tipo estándar que se utiliza para las transmisiones de televisión, o puede ser un dispositivo especializado diseñado específicamente para la visión robótica. Algunos aparatos de esta última clase emplean chips especiales denominados *RAM ópticas*; éstos se componen de memoria RAM en la que el valor de cada byte se establece automáticamente en función de la cantidad de luz que cae sobre ese byte determinado.

Lo que uno cree que ve



¿Cuál es más larga? La ilusión de Muller-Lyer hace que uno crea que la línea vertical de la figura de la izquierda es más larga que la línea vertical de la de la derecha. En realidad, las dos tienen la misma longitud

En la ilusión de la vía, la barra horizontal superior parece ser más larga, independientemente de cómo se mire la figura. En realidad, las dos barras horizontales son de igual longitud

Una figura "imposible", que mezcla en dos dimensiones formas de apariencia válida. La investigación demuestra que los humanos son incapaces de reconocer esto como un objeto

A pesar de que las líneas verticales parecen estar combadas, en realidad son rectas. El ojo inclina la imagen para adaptarla al despliegue de los rayos

Kevin Jones

Los patrones de color ilustrados, tomados de los tests de Ishihara para la ceguera de colores, miden deficiencias de rojo-verde. En el ejemplo de la izquierda, las personas con visión normal verán el número 74, mientras que quienes adolezcan de una deficiencia de rojo-verde, verán el número 21. En el círculo de la derecha, la visión normal no discernirá nada, o sólo un 2 difuso, mientras que quienes padezcan una deficiencia de rojo-verde distinguirán claramente un 2

Estos dispositivos se están abaratando cada vez más y proporcionan una zona de memoria RAM que contiene toda la información relativa a la escena captada por el ojo del robot.

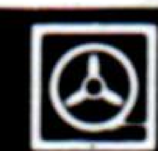
En general, la salida de un ojo robot se retiene en una matriz bidimensional, cada elemento de la cual contiene un valor que corresponde al brillo de la luz que cae sobre esa parte concreta de la escena que se está contemplando. La cantidad de elementos de la matriz proporciona la *resolución* de la imagen, y la gama de números que se puede retener en cada elemento de la matriz determina la cantidad de niveles de la *escala de grises* que se pueden discernir. Tradicionalmente, en los sistemas de visión cada elemento de la matriz se denomina *pixel* o *elemento de imagen*. De modo que una matriz de imagen de 500 por 250 pixels que represente niveles de brillo asignándole un byte a cada pixel tendría una resolución horizontal de 500 pixels, una resolución vertical de 250 pixels, un total de 125 000 pixels y 256 escalas de grises desde el negro al blanco puro. Para dar una idea del detalle que proporcionaría tal imagen, considere una imagen de televisión estándar. El sistema español utiliza 625 líneas verticales, de modo que la resolución vertical es de 625 pixels. Para obtener una resolución similar en sentido horizontal, se necesitarían aproximadamente 1 000 pixels (porque la pantalla es más ancha que alta), y los niveles de escala de grises se podrían representar mediante el mismo único byte para proporcionar 256 niveles de brillo. Un sistema robot con una resolución equivalente daría una imagen aceptable para que la procesara el ordenador.

Los procesos que debe realizar el "cerebro" robot con el objeto de "ver" esta imagen siguen unas pautas establecidas. El primer paso supone ajustar los niveles de escala de grises de modo que los pixels adyacentes con niveles similares de escala de grises se "emparejen" al mismo nivel. El ordenador trabaja sobre la superficie completa de la imagen uniformizando los niveles para eliminar cualquier pequeña irregularidad. Una vez hecho esto, el ordenador vuelve a examinar la imagen, percibiendo todos los pixels adyacentes que poseen niveles de escala de grises notoriamente diferentes; estas diferencias se enfatizan entonces. La finalidad de esto último es lograr que las características importantes de la imagen se marquen con límites, tales como líneas y bordes, que aparezcan como cambios súbitos en el nivel de escala de grises: el ordenador toma nota de los mismos, enfatizándolos para asegurar que sobresalgan.

Después de realizado esto, el ordenador explora de nuevo la imagen en busca de todos los cambios verdaderamente significativos en los niveles de escala de grises. Luego utiliza los mismos de forma muy parecida a como resolvería un ser humano uno de esos acertijos gráficos que aparecen en periódicos y tebeos que consisten en unir correlativamente una serie de puntos para componer una imagen. En la mayoría de estos juegos se cuenta con la ayuda que supone el hecho de que los puntos estén numerados; el ordenador no dispone de tal ayuda y simplemente debe seguir lo que parezca una ruta. Al final de este proceso el robot tendrá una imagen interna de la escena en la cual habrá "suavizado" la imagen y dibujado líneas alrededor de aquellos objetos que parezcan ser importantes.

Pero ¿es esto "ver"? En realidad, todo lo que ha hecho el robot es llevar a cabo ciertas transformaciones de la escena; en otras palabras, todavía no "sabe" qué es lo que está mirando.

Existen dos soluciones para este problema. La primera consiste en programar al robot con un conjunto de reglas que se expresan como una serie de sentencias simples acerca del mundo visual. Esto se conoce como aproximación *de abajo arriba* (*bottom-up*) a la percepción visual, frase que alude al hecho de que el robot empieza con cosas muy simples y a partir de ellas intenta evaluar lo que está viendo en un nivel de comprensión más complejo. El segundo enfoque es el de programar al





Reconocer objetos

Para aprender un objeto, un robot sigue un proceso similar al aprendizaje humano: toma la imagen que ve y la compara con la imagen de un objeto conocido hasta hallar una pareja. A diferencia de los humanos, sin embargo, los robots poseen una capacidad visual limitada y sólo pueden almacenar una gama de patrones de objetos muy restringida. Al no contar con la profundidad de la experiencia humana y las capacidades comparativas tan desarrolladas del hombre, el robot no puede determinar que el objeto percibido corresponde a los perfiles almacenados conocidos como "teléfono", ni siquiera haciendo rotar su imagen en varias posiciones

Kevin Jones

robot con un conjunto de objetos que es probable que vea y dejar luego que examine la imagen para ver si alguno de estos objetos está presente. Esto se denomina aproximación *de arriba abajo* (*top-down*) en razón de que el robot empieza con una idea de alto nivel muy compleja acerca de lo que podría estar viendo y luego comprueba si se corresponde con su verdadera entrada visual.

Para ilustrar la diferencia entre los dos métodos, considere un robot que esté mirando una mesa. La aproximación de abajo arriba consiste en analizar la imagen y descubrir que contiene cuatro partes verticales y, junto a la parte superior de las mismas, una gran superficie horizontal. Esto corresponde al conocimiento preprogramado de que podría haber una gran superficie descansando sobre cuatro patas y que esta estructura sería lo que se denomina una mesa. La aproximación de arriba abajo empezaría con el robot mirando a la mesa e interrogándose: "¿Es esto una mesa?" Esta pregunta se la puede formular porque posee un modelo interno de una mesa con el cual comparar su entrada visual.

En general, la aproximación de abajo arriba capacita al robot para ver cosas con las que nunca se ha encontrado y para entender algo acerca de ellas; pero para hacer esto precisa una gran cantidad de programación detallada que le proporcione las reglas básicas necesarias acerca del mundo con el cual se encontrará. Sin embargo, la aproximación de arriba abajo permite que el robot sólo reconozca objetos sobre los que ya posea algún conocimiento interno; de modo que cualquier cosa nueva supondría problemas.

Los diseñadores de robots utilizan ambos métodos y en ocasiones los combinan. Parece probable que los humanos empleemos procedimientos similares para la percepción visual; no obstante, lo hacemos automáticamente y no somos conscientes de que estos procesos se están realizando.

Pero la visión del robot, de momento, dista mucho de ser perfecta. Ello se debe a varios motivos. Uno de los más importantes es la inmensa cantidad de potencia de proceso que se necesita para procesar una imagen. Recuerde que el sistema de nuestro ejemplo tenía 125 000 pixels almacenados cada uno como un byte: por consiguiente, se deberían procesar más de 122 K de memoria para cada imagen. Si bien hemos simplificado nuestra descripción, muchos de los procesos que se deben llevar a cabo en cada pixel son bastante complejos desde el punto de vista matemático. Si el robot ha de observar el mundo que lo rodea en "tiempo real" (es decir, considerar los acontecimientos a medida que se van produciendo), entonces se reciben 25 imágenes diferentes por segundo (esto también es así en el caso de las cámaras de televisión). Ello significa que el robot necesitaría procesar más de 3 050 K de datos por cada segundo, ¡lo que equivale aproximadamente al contenido de más de una docena de discos flexibles!

Para tratar el problema del proceso se pueden considerar dos enfoques. Uno consiste en desarrollar hardware para fines especiales que efectúe el procesamiento de las imágenes (este tipo de hardware está comenzando a aparecer en la actualidad). Por otra parte, la resolución de la imagen y la cantidad de niveles de escala de grises se podrían reducir para adaptarse al hardware existente. Esto conduciría a que la imagen se procesara más rápidamente, pero la calidad de ésta sería inferior.

En estos momentos, sin embargo, el tema de la visión del robot todavía no se entiende del todo, al menos no más de lo que se comprenden los detalles acerca de la visión del hombre. Cuando utilizan un sistema de visión, los robots a menudo cometen errores. Bien podría suceder que, finalmente, la única respuesta fuera desarrollar sistemas en los cuales el robot "aprendiera" a ver cosas en vez de programarlo detalladamente sobre qué es lo que puede y no puede ver. Y a la larga podría darse el caso de que nunca pudiera "ver" las cosas correctamente hasta el desarrollo de una forma que le proporcionase un conocimiento muchísimo mayor acerca del mundo circundante.

Todas cambian

En el Spectrum es distinta la implementación del programa de sustitución de variables: la utilidad se mezcla al final del programa

A medida que el programa de sustitución de variables explora a través del programa, va haciendo una copia de la versión modificada en una zona por encima de RAMTOP. La versión modificada se vuelve a copiar entonces en la zona del programa principal mediante un programa en código máquina que regula la cantidad de espacio disponible si se ha alterado la longitud del programa, de modo que la nueva versión quepa en la zona para BASIC.

La primera parte del programa en BASIC es similar al programa de búsqueda de variables (véase p. 1145). Hay algunas variables extras, incluyendo Altprog, que señala el comienzo de la zona reservada para la copia del programa, y Altapunt, que lleva el registro de a dónde debe ir el siguiente byte del programa modificado. Los cambios principales implican copiar el programa, en vez de simplemente leerlo. El copiado se realiza mediante la subrutina

Escritura automática

Los opcodes LDIR y LDDR del Z80 son instrucciones para transferencia de bloques que utilizan el incremento o el decremento automáticos: el registro HL se inicializa para apuntar al comienzo del bloque fuente o emisor, DE debe apuntar al comienzo del destino o receptor, y BT debe contener la cantidad de bytes del bloque. LDIR y LDDR copian entonces el byte fuente en el byte de destino, incrementando o decrementando automáticamente HL y DE, y decrementando BC hasta que llega a cero, cuando se considera completa la copia. Observe que LDIR es una copia «tonta» (véase p. 1206): se da por supuesta la inteligencia del programador

HL
DIRECCIÓN FUENTE
FUENTE
BLOQUE
±
BC
DIRECCIÓN DESTINO
DESTINO
BLOQUE
±
DE
MEMORIA
CANT. DE BYTES DEL BLOQUE
−
NO
Es BC = 0 ?
SÍ
Copia completa

Kevin Jones

Programa de sustitución de variables

```
9000 INPUT "Nombre a buscar?"; LINE t$
9005 INPUT "Reemplazar por?"; LINE r$
9010 FOR i=1 TO LEN (t$)
9020 IF t$(i)>="a" AND t$(i)<="z" THEN LET t$(i)=CHR$ (CODE
     (t$(i)) - 32)
9030 NEXT i
9040 LET Distint REM=234
9050 LET Comillas=34
9060 LET Nuevalinea=13
9070 LET Subrayado=95
9080 LET Numero=14
9090 LET PROG=23635
9100 LET Apuntexto=PEEK (PROG)+256*PEEK (PROG+1)
9102 LET Altprog=46000
9105 LET Altapunt=Áltprog
9110 LET Numlinea=256*PEEK (Apuntexto)+PEEK (Apuntexto+1)
9111 PRINT Numlinea
9120 IF Numlinea >=9000 THEN GO TO 9600
9130 LET q=2:GO SUB 9800
9135 LET Longdir=Altapunt
9140 LET Siglinea=Apuntexto+2+PEEK (Apuntexto)+256*PEEK
     (Apuntexto+1)
9150 LET q=2:GO SUB 9800
9160 LET Byte=PEEK (Apuntexto):LET q=1:GO SUB 9800
9170 IF Byte=Nuevalinea THEN GO TO 9110
9180 IF Byte<>Distint REM THEN GO TO 9220
9190 REM Copiar REM sin alterar
9200 LET q=Siglinea-Apuntexto:GO SUB 9800
9210 GO TO 9110
9220 IF Byte<>Comillas THEN GO TO 9280
9230 REM Copiar todo lo que este entre comillas, pero parar al final
     de la linea en caso de comillas sin cerrar
9235 LET q=1
9240 IF PEEK (Apuntexto+q-1)=Nuevalinea THEN GO SUB
     9800:GO TO 9110
9250 IF PEEK (Apuntexto+q-1)=Comillas THEN GO SUB 9800:GO
     TO 9160
9260 LET q=q+1
9270 GO TO 9240
9280 REM Copiar numero binario de 5 bytes
9290 IF Byte=Numero THEN LET q=5:GO SUB 9800:GO TO 9160
9310 REM El primer caracter del nombre debe ser una letra en
     mayuscula o minuscula
9320 IF Byte>=CODE ("A") AND Byte<=CODE ("Z") THEN LET
     c$=CHR$ (Byte):GO TO 9370
9330 REM Usar mayusculas en vez de minusculas
9340 IF Byte>=CODE ("a") AND Byte<=CODE ("z") THEN LET
     c$=CHR$ (Byte-32):GO TO 9370
9360 GO TO 9160
9370 LET n$=""
9380 LET n$=n$+c$
9400 REM Letra, digito o subrayado despues del primer caracter del
     nombre
9410 IF PEEK (Apuntexto)>=CODE ("A") AND PEEK
     (Apuntexto)<=CODE ("Z") THEN LET c$=CHR$ (PEEK
     (Apuntexto)):LET Apuntexto=Apuntexto+1:GO TO 9380
9420 REM Usar mayusculas en vez de minusculas
9430 IF PEEK (Apuntexto)>=CODE ("a") AND PEEK
     (Apuntexto)<=CODE ("z") THEN LET c$=CHR$ (PEEK
     (Apuntexto)-32):LET Apuntexto=Apuntexto+1:GO TO 9380
9440 IF PEEK (Apuntexto)>=CODE ("Ø") AND PEEK
     (Apuntexto)<=CODE ("9") THEN LET c$=CHR$ (PEEK
     (Apuntexto)):LET Apuntexto=Apuntexto+1:GO TO 9380
9450 IF PEEK (Apuntexto)=Subrayado THEN LET c$=CHR$ (PEEK
     (Apuntexto)):LET Apuntexto=Apuntexto+1:GO TO 9380
9460 REM Terminar con $ para variable en serie
9470 IF PEEK (Apuntexto)=CODE ("$") THEN LET n$=n$+"$":LET
     Apuntexto=Apuntexto+1:GO TO 9500
9480 REM (si matriz o funcion
9490 IF PEEK (Apuntexto)=CODE ("(") THEN LET n$=n$+CHR$
     (PEEK (Apuntexto)):LET Apuntexto=Apuntexto+1
9500 IF n$=t$ THEN LET n$=r$
9505 LET Altapunt=Altapunt-1
9510 FOR p=1 TO LEN (n$)
9520 POKE Altapunt,CODE (n$(p))
9530 LET Altapunt=Altapunt+1
9540 NEXT p
9550 IF n$<>r$ THEN GO TO 9160
9560 LET Longbaja=PEEK (Longdireccion)+LEN (r$)-LEN (t$)
9570 IF Longbaja>255 THEN LET Longbaja=Longbaja-256: POKE
     Longdireccion+1,1+PEEK (Longdireccion+1)
9580 POKE Longdireccion, Longbaja
9590 GO TO 9160
```

```
9599 REM Prepararse para desplazar programa alterado nuevamente
     a zona programa principal
9600 LET Antigualong=Apuntexto-(PEEK (PROG)+256*PEEK
     (PROG-1))
9610 LET Nuevalong=Altapunt-Altprog
9620 POKE 45060,Nuevalong-256*INT (Nuevalong/256)
9630 POKE 45061,INT (Nuevalong/256)
9660 POKE 45056,Apuntexto-256*INT (Apuntexto/256)
9670 POKE 45057, INT (Apuntexto/256)
9680 IF Antigualong=Nuevalong THEN RANDOMIZE USR (45084)
9690 IF Antigualong<Nuevalong THEN LET X=Nuevalong-
     Antigualong
9700 IF Antigualong>Nuevalong THEN LET
     X=Apuntexto-(Antigualong-Nuevalong)
9710 POKE 45058,X-256*INT (X/256)
9720 POKE 45059,INT (X/256)
9730 IF Antigualong<Nuevalong THEN RANDOMIZE USR 45062
9740 IF Antigualong>Nuevalong THEN RANDOMIZE USR 45074
9800 FOR p=Apuntexto TO Apuntexto+q-1
9810 POKE Altapunt,PEEK (p)
9820 LET Altapunt=Altapunt+1
9830 NEXT p
9840 LET Apuntexto=p
9850 RETURN
9900 SAVE "SUSTITUCION" LINE 9910:SAVE "SUST MC"CODE
     45064,37: STOP
9910 CLEAR 45055: LOAD "SUST MC"CODE
```

Cargador de código máquina

```
  10 CLEAR 45055
  20 LET a=45062
  30 FOR l=1000 TO 1040 STEP 10
  40 LET s=0
  50 FOR a=a TO a+7
  60 READ b
  70 POKE a,b
  80 LET s=s+b
  90 NEXT a
 100 READ c
 110 IF s<>c THEN PRINT"ERROR DE DATO EN LINEA";l: STOP
 120 NEXT l
1000 DATA 42,0,176,237,75,2,176,205,913
1010 DATA 85,22,24,10,42,0,176,237,596
1020 DATA 91,2,176,205,229,25,33,176,937
1030 DATA 179,237,91,83,92,237,75,4,998
1040 DATA 176,237,176,207,255,0,0,0,1051
```

Programa de sustitución en assembly

```
0000                HCSITI   EQU    $1655
0000                RCLAM1   EQU    $19E5
0000                T0       EQU    $B000
0000                T1       EQU    $B002
0000                T2       EQU    $B004
0000                ALTPRG   EQU    $B3B0
0000                PROG     EQU    $5C53
B006                         ORG    $B006
B006   2A00B0       UP       LD     HL, (T0)
B009   ED4B02B0              LD     BC, (T1)
B00D   CD5516                CALL   HCSITI
B010   180A                  JR     COPY
B012   2A00B0       DOWN     LD     HL, (T0)
B015   ED5B02B0              LD     DE, (T1)
B019   CDE519                CALL   RCLAM1
B01C   21B0B3       COPY     LD     HL, ALTPRG
B01F   ED5B535C              LD     DE, (PROG)
B023   ED4B04B0              LD     BC, (T2)
B027   EDB0                  LDIR
B029   CF                    RST    8
B02A   FF                    DB     $FF
```

de la línea 9800, que copia la cantidad de bytes especificada por q y actualiza los apuntadores de los programas antiguo y nuevo.

Los nombres de las variables se copian mediante el código que comienza en la línea 9500 y, si el nombre de la variable se ha modificado, se alteran los dos bytes del principio de la línea que retienen la longitud de la línea, para reflejar el cambio en la longitud.

Después de que todo el programa se ha modificado y se ha copiado en la nueva zona, el programa BASIC calcula los valores que necesita el código máquina para volver a copiar el programa alterado, y luego coloca (POKE) estos valores en las posiciones de memoria donde el código máquina espera hallarlas. Si el programa nuevo y el antiguo tienen la misma longitud, el nuevo programa se puede copiar en el mismo espacio que ocupa el programa antiguo. En este caso, la única información que precisa el programa en código máquina es la longitud del programa.

Si el programa nuevo es más largo que el antiguo, necesitamos hacer espacio extra en la zona de programas desplazando hacia arriba la rutina de sustitución de variables, que deseamos conservar. El espacio extra se consigue llamando a la subrutina de ROM, HACER-SITIO, en la dirección 1655 hexa. Cuando se llama a HACER-SITIO, el registro doble HL debe contener la dirección de después del lugar donde se ha de hacer el espacio, y el registro doble BC debe contener la longitud del espacio necesario. El valor requerido para HL no es más que el valor final de Apuntexto, y el valor de BC es la diferencia entre la longitud antigua y la nueva.

Si el programa nuevo es más corto que el antiguo, hemos de desplazar hacia abajo el programa de sustitución de variables. Esto lo podemos hacer utilizando la subrutina de ROM RECLAMAR-1 de la dirección 19E5 hexa. Cuando se llama a RECLAMAR-1, el registro doble HL debe contener la dirección del primer byte a dejar solo, y el registro doble DE debe contener la dirección del primer byte a reclamar. El valor requerido para HL es, nuevamente, el valor final de la variable Apuntexto, y el valor requerido para DE se calcula restándole a Apuntexto la diferencia entre la longitud antigua y la nueva.

El programa alterado se vuelve a copiar en la zona del programa principal mediante la instrucción para movimiento de bloques LDIR (*LoaD with Increment and Repeat*). La dirección de comienzo de la zona del programa alterado se carga en HL, la dirección de comienzo de la zona del programa principal en DE, la longitud del programa alterado en BC, y después la instrucción LDIR desplaza todo el programa alterado, byte a byte.

Las dos últimas líneas del programa en lenguaje assembly utilizan otra rutina de ROM, en la dirección 8. Ésta es una rutina de «informe» que imprime un mensaje de error y otros comentarios. La rutina es llamada por la instrucción RST 8, y el informe producido se especifica por el byte que sigue a la instrucción RST 8. El valor del byte es uno menos que el número del informe, de modo que FF hexa, o −1, da el OK o el informe de Programa terminado; 0 da NEXT sin FOR, y así sucesivamente. El programa en código máquina termina con RST8, en vez de la habitual instrucción RET, para evitar el retorno al programa en BASIC que se ha desplazado.

Dando el tono

Veamos otros dos parámetros de la sintetización de sonido: volumen y altura

El volumen de un tono está determinado por la escala de oscilación de la forma de onda que genera el tono. Dicho en otras palabras, el volumen depende de la diferencia entre el valor máximo de la forma de onda y el valor mínimo. Esta propiedad de una onda sonora se denomina *amplitud*.

Utilizando un sencillo programa en BASIC para hacer oscilar valores colocados en el registro de la puerta para el usuario podemos demostrar cuán fácilmente se puede controlar la amplitud de una forma de onda digital.

```
10 REM **** AMPLITUD/FRECUENCIA BAS CBM ****
20 :
25 RDD=56579:REGDAT=56577
30 POKE RDD,255:REM TODAS SALIDA
40 FOR I=255 TO 0 STEP-15
50 FOR J=1 TO 100
60 POKE REGDAT,I:POKE REGDAT,0
70 NEXT J,I

800 REM **** PROGRAMA MUESTRA ****
805 :
806 UP$=CHR$(145)
810 DIV=49798:REM POSICION FACTOR AMPLITUD
820 DEL=49799:REM POSICION FACTOR DEMORA
830 TME=49800:REM FACTOR DURACION
840 CALL=49801:REM DIRECCION COMIENZO PROGRAMA
850 :
860 RDD=56577:POKERDD,255:REM TODAS SALIDA
870 :
880 PRINTCHR$(147):REM LIMPIAR PANTALLA
890 PRINT:INPUT"FACTOR DE AMPLITUD 0-7";FA
900 IF FA<0 OR FA>7 THEN PRINT UP$;UP$;:GOTO890
910 POKE DIV,FA
920 :
930 PRINT:INPUT"FACTOR DE DEMORA 1-101";FD
940 IF FD<0 OR FT>101THEN PRINTUP$;UP$;:GOTO960
980 POKE TME,FT
990 :
1000 SYS CALL
1010 GETA$:IFA$="" THEN 1010
1020 IFA$="X" THEN 880:REM RECOMENZAR
1030 GOTO 1000:REM OTRO BEEP
```

Al principio del programa se genera una forma de onda tosca que oscila entre 255 y 0. Ello significa que la amplitud de la onda es 255. A medida que se va ejecutando el programa, el valor superior colocado en el registro de datos se reduce en pasos de 15. A medida que disminuye el valor superior va disminuyendo igualmente la amplitud; y el efecto de esto, al escuchar el sonido producido a través de un amplificador estéreo o un par de auriculares, es que el volumen del tono se va reduciendo gradualmente hasta desaparecer. De modo que el volumen de un tono sintetizado digitalmente se puede controlar limitando la escala de valores colocada en el registro de datos de la puerta para el usuario.

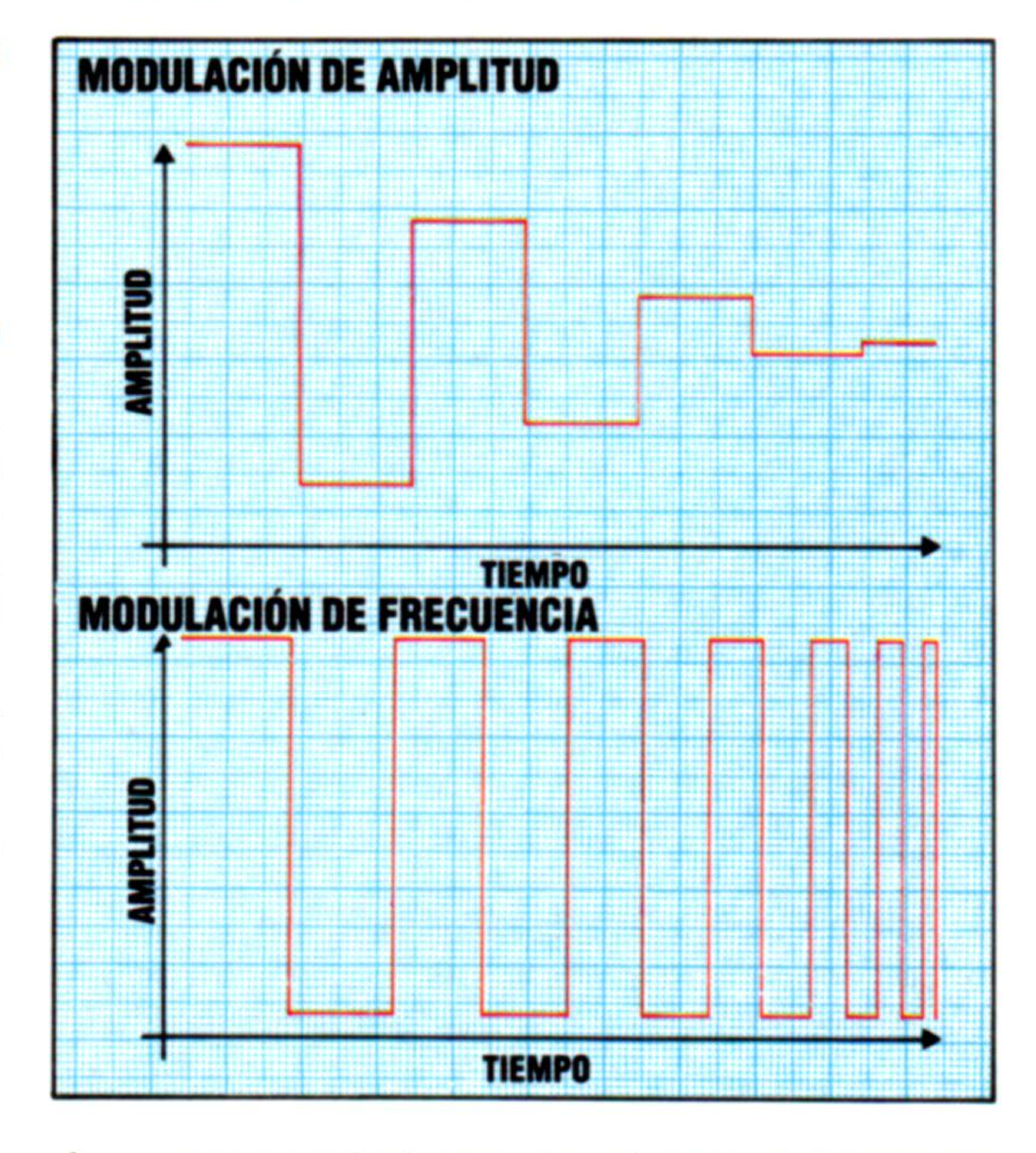
La altura de una nota está gobernada por la frecuencia de la onda generadora; o sea, el número de ciclos de formas de onda por segundo: cuanto mayor es el número de formas de onda producido por unidad de tiempo, más alta es la altura de la nota oída.

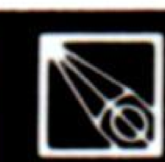
La frecuencia se puede controlar digitalmente de dos maneras principales: la primera es incrementando la frecuencia desde un límite inferior, tomando menos muestras de la forma de onda. Si una onda se dividiera en 100 muestras, por ejemplo, a un programa en código máquina le llevaría un cierto tiempo colocar cada valor en sucesión en el registro de datos, y, por lo tanto, se podría producir cierta cantidad de formas de onda completas por segundo. La cantidad de muestras, obviamente, determina la frecuencia del tono escuchado. Para duplicar la frecuencia, el programa en código máquina podría, en cambio, tomar de la tabla de datos que define la onda sólo un valor de cada dos. La frecuencia se podría triplicar tomando un valor de cada tres, y así sucesivamente. Este método tiene dos desventajas. La primera es que resulta difícil hacer pequeños ajustes de frecuencia sin distorsionar la forma de la onda. En segundo lugar, a medida que la frecuencia aumenta, la onda generada tiene cada vez menos relación con la forma de onda original, dado que se utilizan menos muestras.

Un método alternativo consiste en empezar con un bucle que vaya saltando a través de los datos de la forma de onda lo más rápidamente posible, proporcionando, por lo tanto, una frecuencia máxima. La frecuencia se puede, entonces, ajustar insertando en el bucle pequeñas demoras. Esto nos da un control mucho más preciso sobre la frecuencia del tono, pero significa que el número de muestras que componen la forma de onda debe ser pequeño si se ha de obtener una frecuencia máxima razonablemente elevada. Podemos emplear el segundo de estos dos métodos para producir un programa en código máquina que nos permita controlar tanto la frecuencia como el volumen.

El mejor método de reducir la amplitud de la forma de onda, conservando al mismo tiempo la

Construcción modular
Las formas de onda se pueden modular ya sea por frecuencia o por amplitud. La frecuencia altera la altura del tono oído y está determinada por el número de ciclos de forma de onda producidos por segundo. La amplitud altera el volumen del tono y es la diferencia entre los valores máximo y mínimo de un ciclo. Los diagramas ilustran cómo se puede modular la amplitud de modo que el volumen del tono producido vaya disminuyendo gradualmente y cómo se puede modular la frecuencia para producir un tono cuya altura se eleve





forma global de la onda, consiste en dividir cada valor de la tabla de la forma de onda en función de una constante. Esto se puede hacer de dos maneras: después de cargar en el acumulador cada valor pero antes de colocar el valor en el registro de datos, o antes de entrar en el bucle principal del programa. El primer método aumentará la cantidad de tiempo requerida para ejecutar cada ciclo del bucle principal y, puesto que este factor limita la frecuencia máxima a obtener, debemos optar por el segundo método. A partir de la tabla de forma de onda original se produce una segunda tabla, dividiendo por una constante cada valor tomado de la tabla original, colocando luego el resultado en la nueva tabla. Ésta es utilizada, entonces, por el bucle principal del programa para los datos de forma de onda. El método de división empleado es burdo. Una constante de amplitud determina la cantidad de veces que el valor de la tabla se desplaza hacia la derecha. Puesto que cada desplazamiento hacia la derecha es una división entera por dos, el efecto de utilizar un factor de amplitud de *n* es el de dividir cada tabla por 2*n*.

Cuando el factor de amplitud es cero se utiliza la tabla original, y el programa se modifica a sí mismo para especificar la dirección base de ésta, en vez de la de la tabla de valores divididos.

La ejecución del bucle principal del programa se puede demorar mediante la inserción de un pequeño trozo de código cuya única misión es dejar transcurrir el tiempo mientras se lo ejecuta. Normalmente esto se hace decrementando o bien un número de ocho bits en uno de los registros índice, o un número de 16 bits de la memoria, desde un valor de demora dado hasta cero. Veremos que si calculamos las demoras máximas logradas mediante estos dos métodos, la disminución de un número de ocho bits proporcionará la demora suficiente.

El problema fundamental no es el de proporcionar suficiente demora (es decir, producir la frecuencia más baja), sino proporcionar la demora mínima; dicho en otras palabras, producir la frecuencia máxima. En la página 1212 utilizamos una forma de onda dividida en 80 pasos. El código extra requerido para la demora retrasa tanto el tiempo de ejecución, que ya no resulta práctico tener esta cantidad de pasos. Debajo se indica el código de que consta el bucle principal.

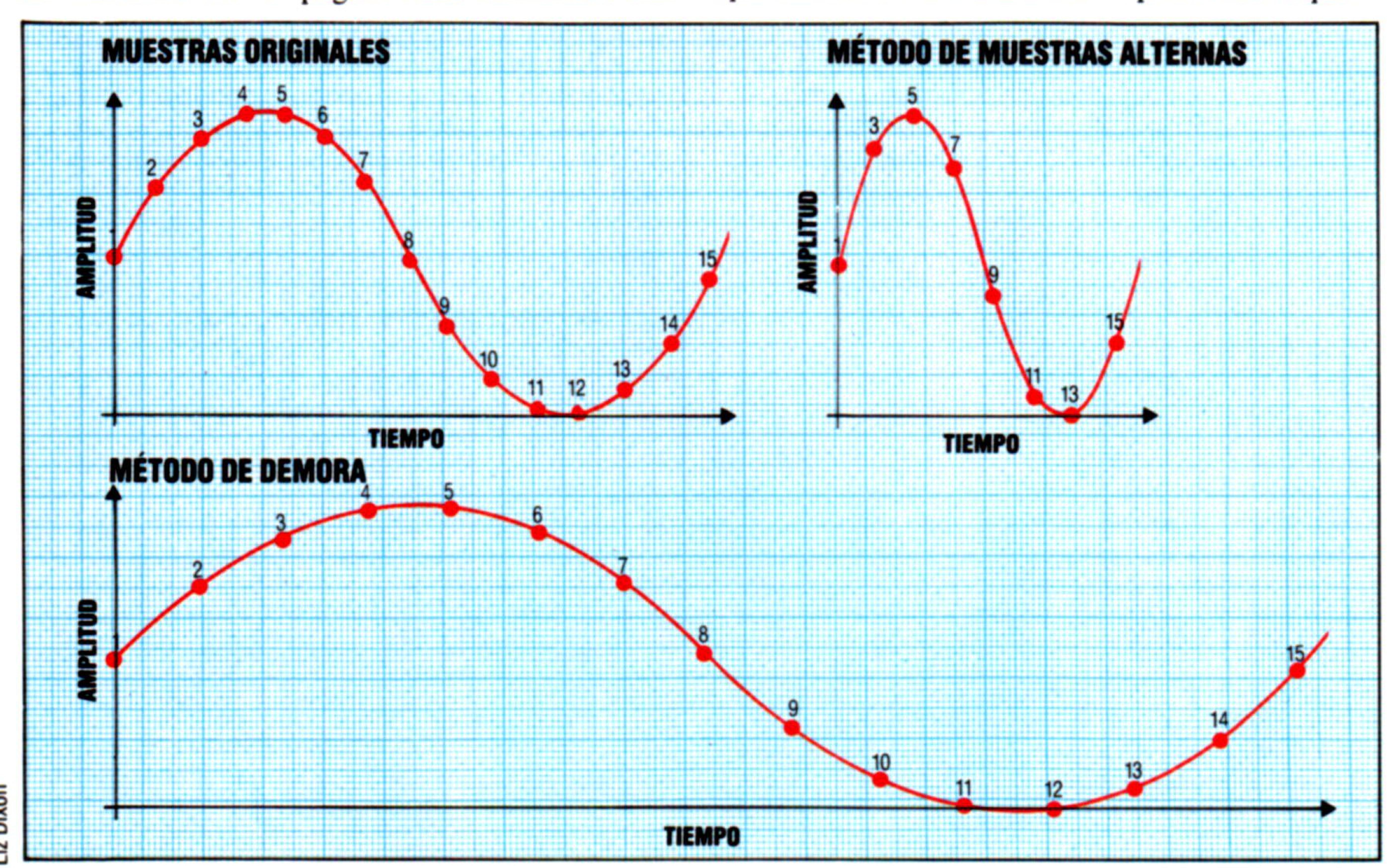
```
PRAL                          Tiempo en ciclos de máq.
          LDX # $00           2
SIGVAL
          LDA AMPTAB,X        4
          LDY DEMORA          4
MASDEM
          DEY                 2
          BNE MASDEM          3 (2 si fracasa el bucle)
          STA PUERTA          4
          INX                 2
          CPX # PASOS         2
          BEN SIGVAL          3 (2 si fracasa el bucle)
```

El número total de ciclos de máquina requeridos viene dado por: 2+(4+4+(2+3)×demora−1+4+2+2+3)×pasos−1=1+(18+5×demora)×pasos: y un valor mínimo de 1 produce la frecuencia máxima: frec máx=1000000÷(1+23×pasos).

Para una frecuencia máxima de alrededor de 3 000 Hz, esta fórmula indica que el número de pasos es 15. Ésta es la cantidad de muestras de la forma de onda que debemos utilizar para producir una frecuencia máxima razonable. Empleando 15 pasos, la fórmula original se puede escribir como: n.º de ciclos de máquina = 271+75×demora.

Si requerimos una frecuencia mínima de, supongamos, 128 Hz (alrededor de dos octavas por debajo de do central), entonces el valor de demora será 101. Este valor puede ser mantenido y decrementado en un registro índice.

El último problema producido por una frecuencia alterada es que, para un número dado de repeticiones del bucle de demora, la duración del tono producido disminuirá a medida que aumente la frecuencia. Ello se debe a que, puesto que ésta aumenta, ejecutar el bucle principal lleva menos tiempo. Para equilibrar esto debemos incluir un cálculo que establezca la cantidad de repeticiones requeri-



Estirando la serpiente
La frecuencia de una onda muestreada digitalmente se puede alterar tomando menos muestras o insertando una demora entre cada valor de muestra. Si la tabla original para la onda contiene 15 muestras de la onda, la frecuencia de la onda de salida se puede duplicar tomando sólo muestras alternas. Por otra parte, se pueden producir todos los valores, los 15, insertando una demora para duplicar el tiempo invertido en producir la onda completa, reduciendo a la mitad la frecuencia. El primer método permite utilizar muchas muestras en frecuencias inferiores pero sólo admite un control burdo de frecuencia. Con el segundo procedimiento es posible lograr un control de frecuencia mucho más fino, pero significa que se pueden utilizar menos muestras

das para producir un tono para una duración dada, independientemente de cuál sea la frecuencia de los tonos. Si hacemos que la unidad de duración sea de 1/50avo de un segundo, el número de repeticiones para un valor de demora dado sería: $1000000 \div (50 \times (271 + 75 \times \text{demora})$. Calcular esto en código máquina sería difícil y lento, de modo que prepararemos desde BASIC una tabla que ofrezca un valor tomado de la fórmula anterior para cada valor de la demora (o sea, de 1 a 109). La rutina en código máquina entonces ha de buscar el valor de repetición y almacenarlo en un decrementador, para producir un tono que dure 1/50avo de segundo.

Los usuarios de Commodore que posean un ensamblador pueden entrar el listado del código fuente y ensamblarlo, para producir un programa objeto que se pueda guardar. El programa de llamada volverá a cargar el programa objeto desde disco o cassette y preparará las tablas. Éntrelo y ejecútelo. Digite NEW y entre el programa muestra en BASIC que da las diversas direcciones de posiciones que utiliza el programa en código máquina. Ejecute este programa con la caja buffer y el convertidor D/A dispuestos tal como se explica en la página 1212. Si no posee ensamblador, entonces entre y ejecute este programa cargador en BASIC antes de ejecutar el programa de llamada. Si utiliza el cargador de BASIC, entonces puede omitir las líneas 45 y 50 del programa de llamada.

Los usuarios de BBC deben entrar la versión para el BBC tal como está escrita y ejecutarla.

```
 10 REM CARGADOR BASIC PARA PROGRAMA FREC/AMP
 20 :
 30 FOR I=49801 TO 49911
 40 READ A:POKEI,A
 50 CC=CC+A
 60 NEXT I
 70 READ CS:IF CC<>CS THENPRINT"ERROR SUMA CONTROL":STOP
100 DATA172,134,194,208,8,169,0,141
110 DATA204,194,76,174,194,169,15,141
120 DATA204,194,162,15,172,134,194,189
130 DATA0,194,24,106,157,15,194,136
140 DATA208,248,202,16,239,120,169,0
150 DATA141,133,194,174,135,194,189,30
160 DATA194,172,136,194,240,7,10,46
170 DATA133,194,136,208,249,141,132
180 DATA194,162,0,189,15,194,172,135
190 DATA194,136,208,253,141,1,221,232
200 DATA224,15,208,239,173,132,194,56
210 DATA233,1,141,132,194,173,133,194
220 DATA233,0,141,133,194,208,218,169
230 DATA0,205,132,194,208,211,88,96
240 DATA16186:REM * SUMA CONTROL *
```

```
 10 REM **** PROGRAMA DE LLAMADA ****
 20 REM ****            Y           ****
 30 REM ****   PREPARACION TABLA   ****
 40 :
 45 DN=8:REM SI CASSETTE DN=1
 50 IF A=0 THEN A=1:LOAD"FREC.HEXA",DN,1
 60 :
 70 REM **** PREPARACION TABLA FORMA ****
 75 :
 80 S=15:TB=12*4096+2*256
 90 FOR I=0 TO S-1
100 Y=127*SIN(X)+127
110 POKE TB+I, Y
120 X=X+2/S
130 NEXT I
140 :
150 REM **** PREPARACION TABLA FREC/DEMORA ****
160 :
170 TB=TB+2*S
180 FOR D=0 TO 101
190 TV=10^6/50*(271+75*D))
200 POKE TB+D,TV
210 NEXT D
```

```
; ++++++++++++++++++++
; ++++++++++++++++++++
; ++                ++
; ++  GENERACION DE ++
; ++  FRECUENCIA Y  ++
; ++    AMPLITUD    ++
; ++     CBM 64     ++
; ++                ++
; ++++++++++++++++++++
; ++++++++++++++++++++
;
PASOS     =15            ;N.º DE PASOS POR CICLO DE ONDA
PUERTA    =56577
;
*=        $C200
SHAPTB    *=*+PASOS      ;TABLA FORMA ONDA
AMPTAB    *=*+PASOS      ;TABLA AMPLITUD
BUCLETB   *=*+102        ;TABLA FREC/DEMORA
CONTADOR  *=*+2          ;CONTADOR BUCLE
DIVISN    *=*+1          ;FACTOR DE DIVISION DE ONDA
DEMORA    *=*+1          ;FACTOR DEMORA
TIEMPO    *=*+1          ;FACTOR DURACION NOTA
;
;**** PREPARAR TABLA AMPLITUD ****
;
          LDY DIVISN
          BNE CONT
          LDA # SHAPTB   ;MODIFICAR PROGRAMA
          STA SIGVAL+1   ;PARA CARGAR SHAPTB
          JMP INIC
;
CONT
          LDA #<AMPTAB
          STA SIGVAL+1
          LDX #PASOS
NEXT
          LDY DIVISN
          LDA SHAPTB,X
MAS
          CLC
          ROR A
          STA AMPTAB,X
          DEY
          BNE MAS
          DEX
          BPL NEXT
;
;**** ESTABLECER VALOR CONTADOR ****
;
INIC
          SEI
          LDA #$00
          STA CONTADOR+1 ;INIC CONTADOR HIBYTE
          LDX DEMORA
          LDA BUCLETB,X
          LDY TIEMPO
          BEQ NOMULT
MULT
          ASL A
          ROL CONTADOR+1
          DEY
          BNE MULT
NOMULT
          STA CONTADOR
;
;**** BUCLE PROGRAMA PRINCIPAL ****
;
PRAL
          LDX #$00
SIGVAL
          LDA AMPTAB,X
          LDY DEMORA
MASDEM                   ;BUCLE DEMORA
          DEY
          BNE MASDEM
;
          STA PUERTA
          INX
          CPX #PASOS
          BEN SIGVAL
;
;**** DECREMENTAR CONTADOR ****
;
          LDA CONTADOR
          SEC
          SBC #$01
          STA CONTADOR
          LDA CONTADOR+1
          SBC #$00
          STA CONTADOR+1
          BNE PRAL
          LDA #$00
          CMP CONTADOR
          BNE PRAL
          CLI
          RTS
```

```
  15 REM ************************************
  20 REM **                                **
  25 REM **        GENERACION DE           **
  30 REM **         FRECUENCIA Y           **
  40 REM **         AMPLITUD BBC           **
  50 REM **                                **
  60 REM ************************************
  90 MODE 7
  95 pasos=15:puerta=&FE60
  97 rdd=&FE62:? rdd=255:REM TODAS SALIDA
 100 HIMEM=HIMEM-&101:REM RESERVAR ESPACIO TABLA
 110 tabla_forma=HIMEM+1
 112 tabla_amplitud=tabla_forma+pasos
 114 tabla_bucle=tabla_amplitud+pasos
 115 PROCpreparar_tablas
 120 PROCcodigo_assemble
 140 REM **** PROGRAMA DE PRUEBA BASIC ****
 160 CLS
 170 PRINT:INPUT"FACTOR DE AMPLITUD 0-7";FA
 180 IF FA<0 OR FA>7 THEN 170
 190 ?factor_div=FA
 200 PRINT:INPUT"FACTOR DE DEMORA 1-101";FD
 210 IF FD<1 OR FD>101 THEN 200
 220 ?factor_demora=FD
 230 PRINT:INPUT"FACTOR DE DURACION 0-15";FT
 240 IF FT<0 OR FT>15 THEN 230
 250 ?factor_tiempo=FT
 265 REPEAT
 270 CALL frec
 280 A$=GET$
 290 UNTIL A$="X"
 300 GOTO 160:REM RECOMENZAR
 900 END
 999 :
1000 DEF PROCcodigo_assemble
1005 DIM MC% &FF
1010 FOR opt%=0 TO 3 STEP 3
1020 P%=MC%
1030 contador=P%:P%=P%+2
1040 factor_div=P%:P%=P%+1
1050 factor_demora=P%:P%=P%+1
1060 factor_tiempo=P%:P%=P%+1
1070 [
1075 OPT opt%
```

```
1080 \**** PREPARAR TABLA AMPLITUD ****
1090
1095 .frec
1100          LDY factor_div
1110          BNE cont
1120          LDA #tabla_forma MOD 256
1130          STA sigval+1
1140          JMP inic
1150\
1160 .cont
1170          LDA #tabla_amplitud MOD 256
1180          STA sigval+1
1190          LDX #pasos
1195 .next
1200          LDY factor_div
1210          LDA tabla_forma,X
1220 .mas
1230          CLC
1240          ROR A
1250          STA tabla_amplitud
1260          DEY
1270          BNE mas
1280          DEX
1290          BPL next
1300 \
1310 \****ESTABLECER VALOR CONTADOR****
1320 \
1330 .inic
1340          SEI
1350          LDA #0
1360          STA contador+1
1370          LDX factor_demora
1380          LDA tabla_bucle,X
1390          LDY factor_tiempo
1400          BEQ nomult
1410 .mult
1420          ASL A
1430          ROL contador+1
1440          DEY
1450          BNE mult
1460 .nomult
1470          STA contador
1480
1490  **** BUCLE PROGRAMA PRINCIPAL ****
1500
1510 .pral
1520          LDX #0
1530 .sigval
1540          LDA tabla_amplitud,X
1550          LDY factor_demora
1560 .masdem
1570          DEY
1580          BNE masdem
1590\
1600          STA puerta
1610          INX
1620          CPX #pasos
1630          BNE sigval
1640 \
1650 \****DECREMENTAR CONTADOR****
1660 \
1670          LDA contador
1680          SEC
1690          SBC #1
1700          STA contador
1710          LDA contador+1
1720          SBC #0
1730          STA contador+1
1740          BNE pral
1750          LDA #0
1760          CMP contador
1770          BNE pral
1780          CLI
1790          RTS
1800 ]
1810 NEXT opt%
1820 ENDPROC
1999 :
2000 DEF PROCpreparar_tablas
2005 x=0
2010 FOR I=tabla_forma TO tabla_forma+pasos-1
2020 y=127*SIN(x)+127
2030 ? I=y
2040 x=x+2*PI/pasos
2050 NEXT I
2060 :
2070 FOR demora=0 TO 101
2080 val_bucle=10^6/50*(271+75*demora))
2090 tabla_bucle?demora=val_bucle
2100 NEXT demora
2120 ENDPROC
```



Dispositivo «mágico»

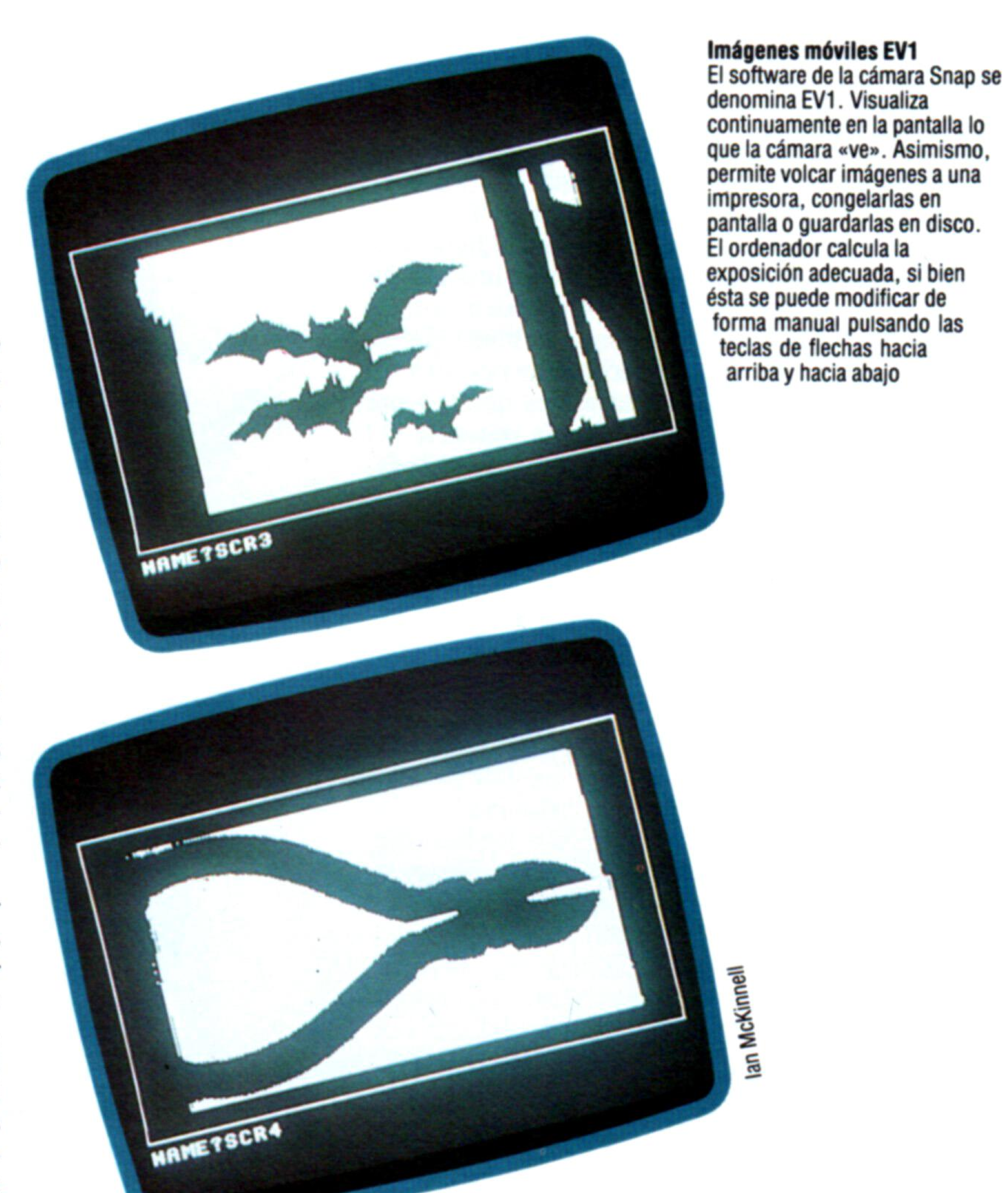
He aquí un dispositivo que permite producir una imagen en pantalla con sólo orientar una cámara hacia el objeto

El EVI Video System, también conocido como "Snap" (instantánea), tiene un precio que se ajusta muy bien al presupuesto del usuario de un ordenador personal. Completo, con el software, cuesta menos de la mitad que cualquier interface para video comparable. El sistema Snap se compone de una pequeña cámara electrónica que se conecta mediante un cable plano a la puerta para el usuario del BBC, junto con software en cassette o bien en disco. Con este sistema se puede transferir la imagen de cualquier objeto o escena a la pantalla de su ordenador simplemente impartiendo la instrucción adecuada. El Snap se puede usar como mera diversión, pero también son posibles aplicaciones más serias: el sistema podría ser la base de una alarma antirrobo "inteligente", o se podría utilizar para reconocimiento de imágenes, quizá incluso confiriéndole "visión" a un robot.

El Snap opera en virtud de un truco de la electrónica. En el interior de la cámara, detrás de la lente, hay un chip de memoria RAM de 32 K. A diferencia de los chips normales, éste se ha fabricado con una ventana transparente en su superficie superior. La imagen de la lente se centra en la superficie del dispositivo de silicio que conforma el chip, en el cual hay una matriz de 256 × 128 diminutas celdas de memoria. Estas celdas, al igual que las de cualquier otro chip, poseen una propiedad que suele ser "invisible" para el usuario. Cuando una celda se expone a la luz, pierde lentamente su carga almacenada, y la velocidad a la cual se descarga es proporcional a la intensidad de la luz que recibe. En la cámara Snap, todas las celdas se cargan primero completamente escribiendo en todas las posiciones de memoria un valor específico, "encendiéndolas" (poniéndolas todas a ON). Al cabo de un breve período, las celdas que reciben luz se descargan lentamente. Después de una pausa, se vuelven a leer para evaluar el valor de cada una. Aquellas que no hayan recibido luz tendrán todavía el mismo valor "encendido", mientras que aquellas que hayan estado expuestas a suficiente luz se habrán descargado, modificando su valor almacenado, y, en consecuencia, se leerán como "apagadas". El software del sistema traza un punto iluminado en la pantalla en una posición que corresponde a cada celda "apagada" de la matriz de memoria. De este modo, se transfiere a la pantalla del BBC una reproducción de la imagen del chip.

El período de tiempo que transcurre entre que el ordenador escribe todas las celdas de memoria para "encenderlas" y las vuelve a leer, determina la "exposición" de la imagen. La cámara Snap tiene capacidad para producir muchas imágenes por segundo si la eliminación resulta suficientemente brillante; con la iluminación normal de una habitación se puede tomar una imagen en menos de un segundo.

La resolución de la imagen está limitada por la matriz de celdas de memoria de 256 × 128 del chip de la cámara. La misma no es particularmente alta (es aproximadamente la misma que la pantalla del BBC en Mode 2 o Mode 5), pero sí proporciona una calidad de imagen muy razonable, comparable a la de la fotografía de un periódico. Una restricción más importante en cuanto a la calidad no es la resolución sino que es consecuencia de otra característica del chip de memoria. La matriz de celdas está, en realidad, dividida en dos segmentos separados en la superficie del chip. Un agujero entre estos dos bloques significa que hay una franja a través del centro de la imagen que la cámara no detecta, y, por lo tanto, la imagen resultante en pantalla tiene una pequeña sección que falta. Sorprendente-



Imágenes móviles EV1
El software de la cámara Snap se denomina EV1. Visualiza continuamente en la pantalla lo que la cámara «ve». Asimismo, permite volcar imágenes a una impresora, congelarlas en pantalla o guardarlas en disco. El ordenador calcula la exposición adecuada, si bien ésta se puede modificar de forma manual pulsando las teclas de flechas hacia arriba y hacia abajo

Ian McKinnell

mente, esto no es demasiado grave y en muchas imágenes pasa totalmente desapercibido.

La cámara Snap se encuentra alojada en una pequeña caja plástica cuyo tamaño es similar al de un paquete de cigarrillos. En su parte frontal está la lente y en la base hay una montura para trípode estándar, que permite estabilizar la unidad cuando se necesitan exposiciones prolongadas. Dos metros de cable plano de ocho vías conectan la unidad con la puerta para el usuario del BBC. Éste es todo el hardware que se requiere.

La cámara utiliza una lente de la pequeña cámara reflex Pentax 110 de una sola lente, instalada en una montura estándar de tipo bayoneta. Tales lentes se pueden adquirir con toda facilidad en tiendas de artículos fotográficos y se venden en una amplia variedad de longitudes focales (se pueden instalar hasta lentes *zoom*), de modo que la lente que se suministra se puede sustituir fácilmente por otra. El único problema de esta disposición es que la lente del sistema Snap está colocada a una distancia del chip que es distinta de la distancia que guardaría, en la cámara Pentax, respecto a la película. Por consiguiente, las marcas de la distancia de enfoque de la lente no tienen ningún significado cuando se la utiliza con el sistema Snap.

La imagen en pantalla producida por la cámara depende en gran medida del software. Con el sistema se suministra un juego de programas y un manual de instrucciones de 50 páginas (preparado de forma muy profesional en un Apple Macintosh) que explica cómo se utiliza el software y los detalles relativos al funcionamiento de la máquina.

La cámara se puede utilizar de dos formas diferentes. Es posible producir imágenes a partir de una única imagen, lo que crea una visualización en pantalla de "dos tonos"; también se puede componer una imagen de "tonos múltiples" tomando varias imágenes con exposiciones distintas. El primer método lo emplea el primer programa del software que se suministra; éste utiliza una pequeña sección de pantalla en Mode 4 para presentar una imagen constantemente actualizada. La exposición se regula pulsando dos de las teclas para el cursor. Este programa incluye, asimismo, una rutina de vuelco de pantalla apta para impresoras Epson o compatibles con la misma.

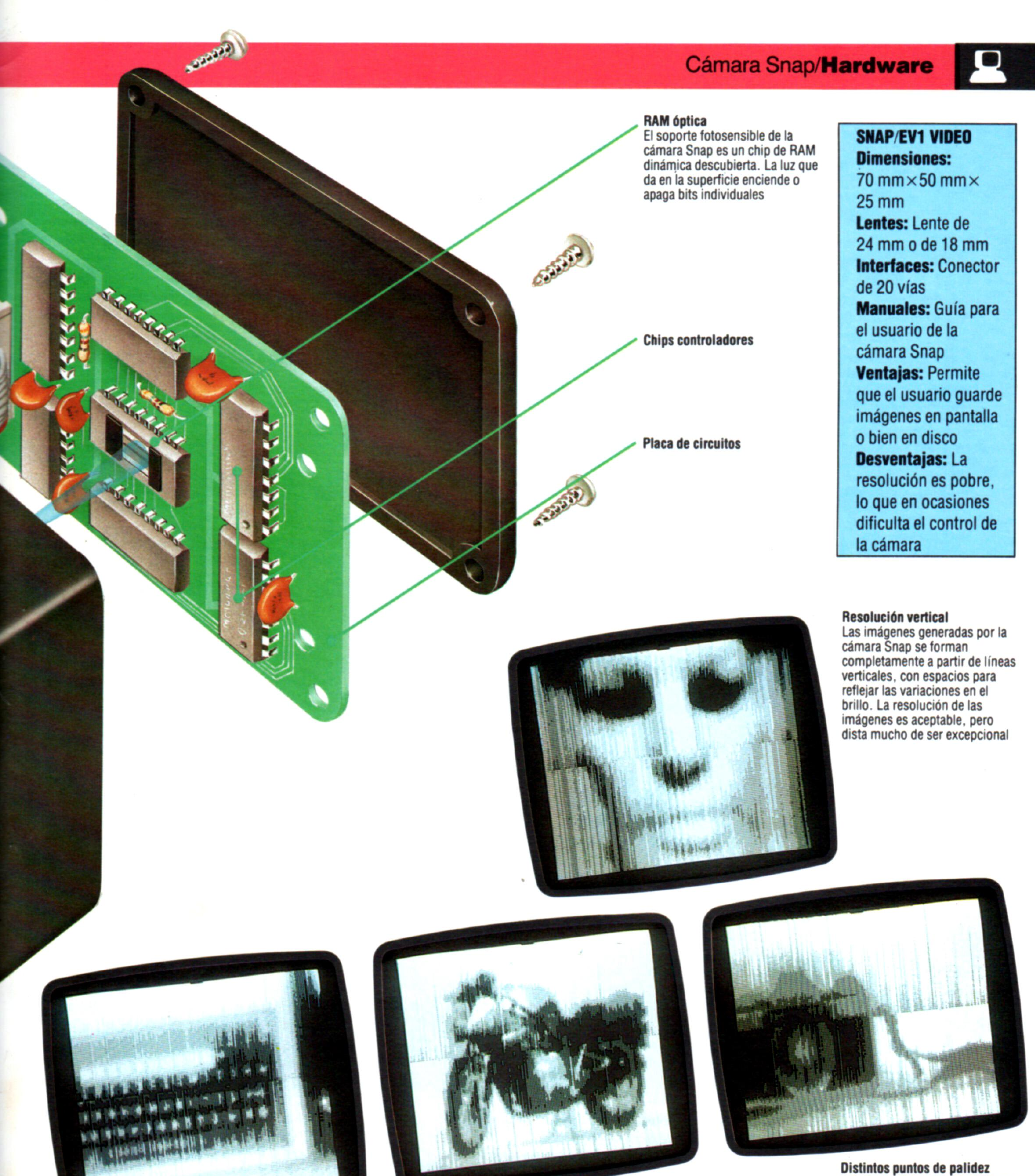
El segundo programa permite construir imágenes más realistas en varios tonos. Se toman ocho imágenes en distintas exposiciones, y éstas se visualizan simultáneamente en una pantalla de Mode 0, utilizando el sombreado para dar más énfasis a las imágenes de exposición corta. El efecto de esto es una imagen que contiene ocho grados distintos de brillo, de negro a blanco pasando por diversas tonalidades de gris. Esto da un efecto más realista, pero a la luz normal de una habitación la producción de estas imágenes en "escala de grises" puede llevar hasta 10 segundos: no del todo una "instantánea".

El programa *Secure* convierte su sistema Snap en una alarma antirrobo informatizada. El software toma nota sólo de las diferencias entre imágenes sucesivas, de manera que puede colocarse de forma que produzca una señal de alarma cuando estas diferencias alcancen un determinado nivel. Por ejemplo, si la cámara está orientada hacia el exterior de su casa, ignorará el hecho de que los árboles sean mecidos por el viento, pero disparará la alarma si llega un visitante a la puerta de entrada.



Movie almacena una corta secuencia de fotogramas de dos tonos, que se reproducen luego rápidamente, creando un efecto de animación. También se suministra un programa llamado *Animal*. Éste es una versión en video del conocido juego por ordenador llamado *Animales* (véase p. 732). Para jugar a él el usuario debe presentar al sistema una imagen que contiene diversos objetos. El programa analiza la imagen, tomando nota de los perfiles de los distintos objetos que "ve", y luego invita al jugador a dar los nombres de cada uno. Si entonces el usuario dirige la cámara hacia otra imagen o escena, ésta intentará identificar cualquiera de los objetos que también hayan aparecido en la primera imagen.

Para los usuarios de ordenadores personales, tal vez la opción más atractiva sea el programa denominado *Arty*. Éste se utiliza para producir complejas imágenes en pantalla a partir de otras distintas. Se emplea la pantalla de Mode 1 completa y las imágenes captadas por la cámara se pueden posicionar en cualquier lugar de la pantalla, con tamaño ampliado o reducido, mediante el empleo de



SNAP/EV1 VIDEO
Dimensiones: 70 mm×50 mm×25 mm
Lentes: Lente de 24 mm o de 18 mm
Interfaces: Conector de 20 vías
Manuales: Guía para el usuario de la cámara Snap
Ventajas: Permite que el usuario guarde imágenes en pantalla o bien en disco
Desventajas: La resolución es pobre, lo que en ocasiones dificulta el control de la cámara

Resolución vertical
Las imágenes generadas por la cámara Snap se forman completamente a partir de líneas verticales, con espacios para reflejar las variaciones en el brillo. La resolución de las imágenes es aceptable, pero dista mucho de ser excepcional

Distintos puntos de palidez
Las imágenes de la cámara Snap normalmente se visualizan en blanco y negro. Un programa de escala de grises denominado *Grey* permite visualizar una imagen a través de toda la pantalla en ocho niveles de brillo. La acrecentada capacidad para reflejar contrastes de tonalidades aumenta el realismo de la imagen visualizada

una palanca de mando. Los colores de primer plano y fondo se pueden cambiar utilizando las teclas de función. Para aprovechar al máximo este programa se necesita cierta dosis de paciencia, pero a partir de unos cuantos objetos de la vida real se pueden construir complejas pantallas a todo color.

Si bien el software que se suministra es complejo y está bien elaborado, cubre una gama de aplicaciones limitada. El programa de imágenes en escala de grises, por ejemplo, sólo se puede emplear en Mode 0. Sería más útil una opción para hacerlo en Mode 2, usando como tonos de gris los distintos colores disponibles en esa modalidad. En el manual se ofrece una gran cantidad de información relativa a las rutinas en código máquina que utiliza el software, pero para hacer uso de las mismas se requiere conocer dicho lenguaje. El usuario de BASIC queda limitado al software suministrado.

Haciendo planes

Esta vez analizaremos el «Graph Plan», un paquete combinado de hoja electrónica y gráficos creado para el BBC Modelo B

A diferencia de los otros paquetes que hemos examinado en esta serie, *Graph Plan* es un programa basado en disco. Acorn regala el paquete, como parte de un conjunto de software gratuito, a los compradores del segundo procesador Z80 (necesario para el uso del paquete). Al igual que todos los programas de este lote, *Graph Plan* es un paquete muy útil y fiable.

El programa no tiene tanto "estilo" como el *Abacus* de Psion, el paquete de hoja electrónica/gráficos que reciben sin cargo alguno los usuarios del Sinclair QL (véase p. 1204). La ventaja sobre el *Abacus* reside en las rápidas velocidades de acceso y almacenamiento de que disfruta el software basado en disco. (Los microdrives del QL son adecuados si se está acostumbrado al software basado en cassette, pero resultan de una lentitud frustrante si uno está habituado a utilizar una verdadera unidad de disco.) Puesto que *Graph Plan* es un paquete "de regalo", es seguro que gozará de popularidad, aunque no tanto, tal vez, como el *Abacus*, porque el precio combinado del procesador Z80 más las unidades Acorn es muy superior al del QL.

En este capítulo concentraremos nuestra atención en la vertiente de gráficos de los modelos financieros. *Graph Plan*, como su nombre sugiere, posee unas capacidades para gráficos muy amplias, así como una formidable matriz de fórmulas comerciales y matemáticas incorporadas.

Lo que le confiere al *Graph Plan* su inusual estilo es su original forma de comunicarse mediante números; toda la interacción del usuario con el programa se realiza a través de 144 instrucciones numeradas. Al cargarlo, el programa tiene una visualización de hoja electrónica estándar (dividida en filas y columnas) que ocupa la mayor parte de la pantalla. En todo el lado derecho de ésta se visualizan las 20 instrucciones básicas con sus números. El usuario selecciona una instrucción y digita el número cuando así se le solicita en la tercera línea de estado, encima de la visualización, mediante el mensaje ENTER COMMAND (entre instrucción).

A pesar de que seleccionar una instrucción (ya sea desde el menú en pantalla o bien de la lista completa de instrucciones del excelente manual del *Graph Plan*, de 124 páginas) es suficientemente sencillo, esta forma de hacer las cosas tiene desventajas obvias. La mayor parte de los paquetes de modelos, en especial los que han obtenido tanto éxito, como el *Lotus 1-2-3* (véase p. 1124), exigen que el usuario entre solamente la letra inicial de una instrucción (o que la seleccione en una visualización mediante las flechas del cursor).

Los paquetes de modelos sofisticados, como el *Lotus 1-2-3*, visualizan explicaciones sobre la función de cada instrucción. *Graph Plan*, por el contrario, da por sentado que el usuario comprende la función de todas sus instrucciones. No obstante, el programa sí proporciona la facilidad para alterar la lista de instrucciones a la derecha de la pantalla, de modo que el usuario puede confeccionar la lista más útil para él. Si, por ejemplo, se selecciona la instrucción número 2 data, el menú de instrucciones pasa a visualizar las instrucciones de la 29 a la 48 (las instrucciones para entrada y manipulación de datos). Existe asimismo una facilidad HELP (ayuda) (instrucción número 7) que ofrece la explicación de la función de la instrucción dada.

Además del sistema de instrucciones numeradas, *Graph Plan* posee otras características exclusivas. La mayoría de las hojas electrónicas, por ejemplo, se basan en el concepto de "celda": una intersección entre una fila y una columna. Graph Plan trata las filas y las columnas como entidades separadas, y el "puntero de datos" es un indicador de cursor que, además de visualizar la identidad del cuadrado al que está apuntando el cursor, indica si se está en modalidad de fila o de columna.

Esta distinción carecería de significado en un sistema en el cual la dirección de celda individual fuera el punto central de referencia, pero en el *Graph Plan* reviste una enorme importancia, porque los gráficos se dibujan con referencia a filas o bien a columnas, pero no a ambas. Para indicar al paquete si se desea generar un gráfico basado en fila o basado en columna, debe establecerse la modalidad correcta cambiando el puntero de datos. Esto se realiza pulsando las teclas con flecha, para desplazar el cursor hasta el encabezamiento de una fila o de una columna, lo cual automáticamente establece la modalidad correspondiente.

Un modelo sencillo

A modo de ilustración de la técnica para dibujar gráficas del paquete, vamos a considerar un modelo sencillo. Posee cinco columnas, rotuladas de "Enero" a "Mayo", respectivamente, y cinco filas, denominadas "Ventas", "Costes de ventas", "Beneficios brutos", "Gastos generales" y "Beneficios netos". En un modelo como éste, una gráfica basada en filas tendrá un significado diferente al de una gráfica basada en columnas. Por ejemplo, podríamos generar un gráfico de barras muy sencillo basado en filas para las cifras del movimiento total de ventas para los meses de enero a mayo.

El gráfico visualizaría los títulos de las columnas (de "Enero" a "Mayo") a lo largo del eje x, y las barras representarían los valores dados en la fila uno del modelo. Por el contrario, situando el puntero de datos en la modalidad de columna, podríamos generar una gráfica de barras muy distinta. Ésta visualizaría los títulos de las filas ("Ventas", "Coste de ventas", etc.) a lo largo del eje x, con las barras representando los valores dados en la columna uno ("Enero") del modelo.

Con los datos apropiados, *Graph Plan* puede producir instantáneamente una cantidad considerable de gráficos diferentes a partir de los datos suministrados en este modelo. Todos estos gráficos se pueden ver, a su vez, en la pantalla, sin ninguna intervención por parte del usuario. *Graph Plan* permite hacer esto mediante la utilización de la instrucción de gráficos 62 SELECT. Entrando 62 en respuesta al mensaje ENTER COMMAND se genera otro mensaje en la línea de estado tres. Si usted se halla en "modalidad fila", por ejemplo, se le solicitará que dé el número de la fila de datos que desea representar gráficamente. Apenas acaba de seleccionar la fila, el mensaje cambia y pasa a solicitarle qué tipo de gráfico desea: "Choose (Bar=1, Line=2, Pie=3)" (Elija: De barras=1, De línea=2, De tarta=3). Entonces puede contemplar inmediatamente el gráfico seleccionando la instrucción 61 DISPLAY. Y puede pasar de gráficos de barras a gráficos de líneas o en forma de tarta a voluntad.

Si desea más flexibilidad en el trazado y el diseño de un gráfico, puede emplear la instrucción 63 OPTIONS. El manual visualiza un "diagrama de decisión" muy bueno, que ilustra claramente el proceso de selección que ha de seguirse cuando se utiliza esta instrucción. La instrucción 63 le presenta en pantalla un menú de seis opciones: Display Chart, Define Chart Options, Define Axes Options, Define Pie Options, Print Chart y Plot Chart (visualizar diagrama, definir opciones del diagrama, definir opciones de eje, definir opciones de tarta, imprimir diagrama y trazar diagrama). Además, el manual posee un apéndice especial, titulado "Guía de los submenús para gráficos", que es de lectura obligada para el usuario novel. En él se explica sucintamente cómo se pueden agregar títulos en un gráfico, seleccionar las tonalidades o el color de gráficos de barras, y realizar toda clase de variaciones de escala (hasta hacer los ejes logarítmicos en vez de lineales).

Una buena facilidad para gráficos incorporada en forma de hoja electrónica (que posee funciones matemáticas y estadísticas) hace que el *Graph Plan* sea apto para una amplia gama de aplicaciones científicas y de ingeniería relativamente sencillas. El paquete proporciona un medio excelente para presentar datos, ya sea para informes o para conferencias, y, por consiguiente, les agradará tanto a los técnicos y a los científicos como a los usuarios comerciales.

Gráfico de barras

Modelo y menú de hoja electrónica

Gráfico de línea

Gráfico de tarta

April 24%

May - 30%

Janua 11%

February - 15%

March - 20%

Ian McKinnell

Visualización de hoja electrónica
Graph Plan se diferencia de las otras hojas electrónicas en dos aspectos. El primero de ellos es el menú de instrucciones numeradas, a la derecha de la pantalla. El segundo es la capacidad de tomar datos de la hoja electrónica y presentarlos en forma gráfica. Graph Plan puede visualizar datos de una fila o de una columna en tres formatos diferentes, como vemos en las fotografías

Poesía en movimiento

En este capítulo centraremos nuestra atención en el tratamiento de listas, fundamental para la forma en que opera este lenguaje

Una lista es un conjunto ordenado de objetos, y en LOGO se identifica mediante la utilización de corchetes; de modo que [CEILAN MADRAS VINDALOO] es una lista. En esta serie ya nos hemos encontrado en varias ocasiones con listas. De hecho, en LOGO no podemos prescindir de ellas, porque es un lenguaje basado en listas. Ya hemos visto cómo una definición para un cuadrado (REPEAT 4 [FD 50 RT 90]) posee una lista de instrucciones (encerradas entre corchetes) como su segunda entrada. Del mismo modo, MAKE "ENT REQUEST le asigna a ENT una lista compuesta por la entrada desde el teclado.

Se pueden asignar listas a variables locales: por ejemplo, MAKE "CURRY [CEILAN MADRAS VINDALOO]. La instrucción PRINT :CURRY imprime la lista sin los corchetes: CEILAN MADRAS VINDALOO.

En LOGO, un objeto puede ser un número, una palabra o una lista, y esta última se define simplemente como un conjunto de objetos. Ésta, por supuesto, es una definición recursiva; una lista puede contener otra lista, o una lista de listas, y así sucesivamente. [[POLLO TIKKA] NAN ENSALADA] es una lista válida, teniendo como primer elemento una lista ([POLLO TIKKA]). Los procedimientos recursivos suelen ser necesarios para procesar listas, precisamente porque éstas son objetos recursivos.

La mayor parte de nuestra programación en LOGO ahora se ha referido a un número o una palabra cada vez. Cuando deseamos procesar grupos de objetos simultáneamente, necesitamos organizar estos objetos simples en una única unidad. El LOGO tiene en la lista su método básico para agrupar objetos simples. La elección de ésta se debe a que es muy versátil: partiendo de una lista se puede crear cualquier organización compleja de datos.

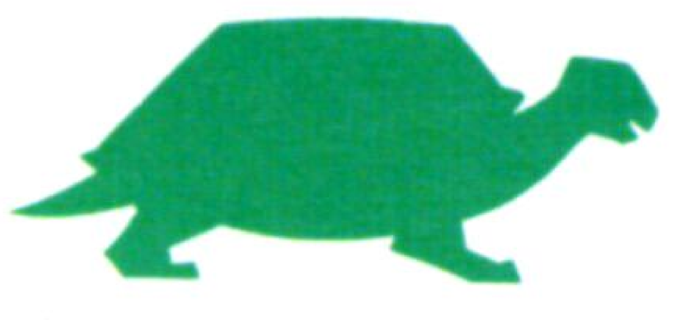
Las dos operaciones de lista fundamentales son FIRST y BUTFIRST. FIRST [CEILAN MADRAS VINDALOO] produce CEILAN, es decir, nos da el primer elemento de la lista. BUTFIRST [CEILAN MADRAS VINDALOO] produce MADRAS VINDALOO; en otras palabras, nos la proporciona sin su primer elemento.

He aquí un procedimiento que imprime los elementos de la lista, uno debajo del otro:

```
TO IMPRIMIR :LISTA
   PRINT FIRST :LISTA
   IMPRIMIR BUTFIRST :LISTA
END
```

De modo que IMPRIMIR [CEILAN MADRAS VINDALOO] nos da:

```
CEILAN
MADRAS
VINDALOO
```



La primera instrucción imprime el primer elemento y después pasa la tarea de imprimir el resto de la lista de entrada a otra copia del procedimiento IMPRIMIR. Cuando ejecute este procedimiento, al acabarse los datos obtendrá un mensaje de error. Ésta es una forma más elegante de terminar:

```
TO IMPRIMIR :LISTA
   IF EMPTY? :LISTA THEN STOP
   PRINT FIRST :LISTA
   IMPRIMIR BUTFIRST :LISTA
END
```

EMPTY? (¿vacía?) verifica si su entrada es la "lista vacía": []. Algunas versiones MIT no poseen la primitiva EMPTY?, pero ésta puede definirse así:

```
TO EMPTY? :LISTA
   IF :LISTA = []THEN OUTPUT "TRUE
   OUTPUT "FALSE
END
```

Similares a FIRST y BUTFIRST son LAST y BUTLAST. LAST [CEILAN MADRAS VINDALOO] produce VINDALOO, y BUTLAST [CEILAN MADRAS VINDALOO] produce CEILAN MADRAS.

Para nuestra primera exploración del tratamiento de listas, intentaremos imitar algunos balbuceos al azar en el diván del psicoanalista. Primero le asignaremos a la variable PALABRAS todos los vocablos que conocemos:

```
MAKE "PALABRAS [MADRE PADRE SEXO
ASESINATO CELOS FUEGO MAR MUERTE SUEÑO]
```

Queremos producir un flujo constante y al azar de estas palabras, porque la experiencia nos ha demostrado que son éstos los vocablos que siempre resultan útiles para captar la atención del psicoanalista.

Para obtener un elemento aleatorio necesitamos seleccionar un número al azar, n, entre uno y la longitud de la lista (nueve, en este caso) y luego seleccionar el enésimo elemento de ésta.

```
TO ENE :NO :LISTA
   IF :NO=1 THEN OUTPUT FIRST :LISTA
   OUTPUT ENE :NO-1 BUTFIRST :LISTA
END
```

Vamos a utilizar este procedimiento en algunos ejemplos, para ver cómo funciona. Supongamos que usted digita ENE 1 :PALABRAS. La condición de la primera línea es verdadera, de modo que el procedimiento produce FIRST :PALABRAS, que en nuestro ejemplo es MADRE.

Pruebe con ENE 2 :PALABRAS: ahora la condición es falsa, de modo que el procedimiento produce ENE 1 BUTFIRST :PALABRAS. Éste ignora el primer elemento de la lista y toma la primera palabra del resto de la misma: PADRE.

De manera que nuestro procedimiento para imprimir al azar una palabra de nuestro limitado vocabulario sería:

```
TO TOMARALAZAR :LISTA
   OUTPUT ENE ((RANDOM 9)+1) :LISTA
END
```



Para utilizarlo, digite TOMARALAZAR :PALABRAS.

Nuestro procedimiento está restringido a listas de nueve elementos. Podríamos mejorarlo si pudiéramos determinar cuántos hay en una lista determinada. Éste es un procedimiento que realiza eso:

```
TO LONGITUD :LISTA
  IF EMPTY? :LISTA THEN OUTPUT 0
  OUTPUT 1 + LONGITUD BUTFIRST :LISTA
END
```

Para ver cómo funciona, pruebe con: LONGITUD [CIENCIA FICCION]. Como la lista contiene algunas palabras, la condición fracasa, de modo que el procedimiento produce 1 + LONGITUD [FICCION]. Ahora LONGITUD [FICCION] produce 1 + LONGITUD []. Al llamar a LONGITUD con una entrada de [], la condición de la línea 1 es verdadera, por lo que el procedimiento produce 0. Ahora LONGITUD [FICCION] produce 0 + 1 = 1 y finalmente LONGITUD [CIENCIA FICCION] produce 1 + 1 = 2. Por lo tanto, un procedimiento más general para tomar palabras al azar de una lista de cualquier longitud sería:

```
TO TOMARALAZAR :LISTA
  OUTPUT ENE ((RANDOM LONGITUD :LISTA)
  +1) :LISTA
END
```

Muchas versiones de LOGO poseen una primitiva, ITEM, que hace exactamente lo que hace ENE, y una primitiva denominada COUNT que realiza lo mismo que LONGITUD. Utilizando estas primitivas, podemos reescribir el procedimiento:

```
TO TOMARALAZAR :LISTA
  OUTPUT ITEM ((RANDOM COUNT :LISTA)
  +1) :LISTA
END
```

Para imprimir una selección de 10 términos que hagan que su psicoanalista se mantenga atento, simplemente digite:

```
REPEAT 10 [PRINT TOMARALAZAR :PALABRAS]
```

Existe un patrón para estos programas procesadores de listas que han compartido muchos de nuestros procedimientos repetitivos para gráficos tortuga. Este patrón es:

- Si la tarea a realizar es muy sencilla, entonces hágala y deténgase.
- De lo contrario, efectúe una pequeña parte de la tarea.
- Después pase el resto de la tarea a otro procedimiento (con frecuencia una copia del procedimiento original).

Existe una estrategia que ofrece elevados porcentajes de éxito, con la cual nos encontraremos reiteradamente en los programas de tratamiento de listas. Compare este programa para dibujar polígonos:

```
TO POLI :N
  IF :N = 0 THEN STOP
  FD 30 RT (360/:N)
  POLI :N - 1
END
```

con la versión de IMPRIMIR que ofrecimos anteriormente. La estructura de ambos procedimientos es idéntica.

Poesía aleatoria

Después de no haber logrado impresionar a nuestro psicoanalista, ahora dedicamos nuestro esfuerzo a la poesía. Aquí produciremos frases completas en lugar de palabras individuales.

```
TO POEMA1 :LONGITUD
  IF :LONGITUD = 0 THEN PRINT "STOP
  (PRINT1 "' ' TOMARALAZAR :PALABRAS)
  POEMA 1 :LONGITUD - 1
END
```

PRINT1 "' ' se incluye para imprimir un espacio entre cada palabra. Para utilizar este procedimiento digite POEMA1 6 para una frase de seis vocablos.

Sería útil poder ampliar nuestra lista original de palabras sin tener que tomarnos el trabajo de volverla a escribir entera. Una manera de lograrlo es a

Abreviaturas

BUTFIRST	BF
BUTLAST	BL
SENTENCE	SE



Sir John Tenniel

Canta la Tortuga Artificial
He aquí un extracto de la canción de la Tortuga Artificial, personaje del famoso cuento *Alicia en el País de las Maravillas*, de Lewis Carroll. El patrón de la métrica, un poco difícil de obtener en una poesía generada por ordenador, está adaptado de una antigua canción popular de origen negro

"¿Por qué no vas más aprisa?", le dijo una pescadilla
a un caracol.
"Tras nosotros viene, muy cerca, un delfín pisándome
la cola.
¡Fíjate cuán raudas las langostas y las tortugas
avanzan todas!
Están esperándonos sobre el cascajo. ¿No querrás venir
y bailar también?"
Querrás, querrías, querrás, querrías,
¿no querrás tú bailar también?
Querrás, querrías, querrás, querrías,
¿no querrías tú también bailar?

"No sabes, no puedes saber, cuán agradable es el
vaivén
cuando levantándonos nos arrojen con las langostas
¡hacia el mar!"
Pero el caracol respondía: "¡Muy lejos! ¡Demasiado lejos!",
y ni se dignaba mirar a dónde.
Dijo que le agradecía a la pescadilla la invitación,
pero que al baile no se uniría.
No querría, no podría, no querría, no podría,
no querría bailar también.
No querría, no podría, no querría, no podría,
no podría también bailar.

través de la utilización de la operación SENTENCE (frase), que toma dos entradas y elabora con ellas una lista. De modo que SENTENCE "MERMELADA [JARRA MIEL] produce [MERMELADA JARRA MIEL].

```
TO AGREGARPALABRAS1 :LISTA
   MAKE "PALABRAS SENTENCE :LISTA :PALABRAS
END
```

De manera que ahora podemos ampliar PALABRAS mediante AGREGARPALABRAS [ANSIEDAD REPRESION [MIEDO A VOLAR]]. Pero se nos presenta el problema de que a la variable PALABRAS no se le ha asignado previamente ningún valor. Para obviar este contratiempo se utiliza la primitiva THING?, que verifica si a una variable se le ha asignado algún valor; produce verdadero (*true*) si su entrada tiene algún valor asignado. Ahora podemos incrementar nuestra lista mediante AGREGARPALABRAS1:

```
TO AGREGARPALABRAS1 :LISTA
   IF NOT THING? "PALABRAS THEN MAKE
   "PALABRAS []
   MAKE "PALABRAS SENTENCE :LISTA :PALABRAS
END
```

Empleando una lista de palabras diferente, obtuvimos la siguiente "poesía" aplicando este procedimiento:

```
APARICION RUIDOSAMENTE HABLO ESPLENDIDO
PARANOICO PLANETA ATERRORIZADO LA CON
VERDE APARICION FLOTANDO PARANOICO ROBOT
HOMBRE VOLO HABLABA FLOTANDO
RUIDOSAMENTE
```

Uno de los defectos más evidentes de nuestra poesía informatizada es su ignorancia total de la gramática. Los poemas podrían tener más sentido si pudiésemos ceñirnos a algunos patrones sintácticos simples, tales como: sustantivo, verbo, sustantivo. Una forma de hacer esto consiste en tener varias listas, una para cada parte de la oración. Entonces podríamos elegir una palabra de cada lista según la estructura que deseáramos para la frase.

Vamos a dejar este problema para que lo investigue usted mismo. En el próximo capítulo del curso le enseñaremos algunas maneras de mejorar las aptitudes de la tortuga para escribir poesía.

Complementos al LOGO

Algunas versiones de LOGO MIT no poseen EMPTY?, ITEM ni COUNT. En todas las versiones LCSI utilice:

EMPTYP por EMPTY?
LISTP por LIST?
TYPE por PRINT1

Hay una primitiva, EQUALP, que verifica si sus dos entradas son la misma. Empléela para comparar listas y palabras en lugar del signo (=). (El signo de igualdad funciona para listas en algunas versiones LCSI, pero en otras no.)

Recuerde la sintaxis diferente de IF:
IF EMPTYP :LISTA [OUTPUT 0]
En el LOGO Atari utilice SE por SENTENCE, y tenga en cuenta que ITEM no está implementado

Ejercicios

1) Escriba un procedimiento para imprimir una lista por orden inverso (utilice LAST y BUTLAST). Modifíquelo para que produzca la lista invertida

2) Escriba un procedimiento que elimine un elemento de una lista. DELETE "COMIDA [BEBIDA COMIDA] produce [BEBIDA] y DELETE "VINO [BEBIDA COMIDA] produce [BEBIDA COMIDA]

Respuestas a los ejercicios

Respuestas a los ejercicios de p. 1217:

1. Cálculo de potencias:

```
TO POTENCIA :A :N
   IF NOT ((INTEGER :N)=:N) THEN PRINT [SOLO
   INDICES DE NUMEROS ENTEROS] STOP
   IF :N=0 THEN OUTPUT 1
   OUTPUT :A* POTENCIA :A :N – 1
END
```

2. Conversión a hexadecimal:

```
TO IMPRESION.HEXA :N
   IF :N < 10 THEN OUTPUT :N
   IF :N = 10 THEN OUTPUT "A
   IF :N = 11 THEN OUTPUT "B
   IF :N = 12 THEN OUTPUT "C
   IF :N = 13 THEN OUTPUT "D
   IF :N = 14 THEN OUTPUT "E
   IF :N = 15 THEN OUTPUT "F
END

TO HEXA :N
   IF :N = 0 THEN STOP
   HEXA QUOTIENT :N 16
   PRINT1 IMPRESION.HEXA REMAINDER :N 16
END
```

3. Comprobar si un número es par:

```
TO PAR? :N
   IF ((REMAINDER :N 2)=0)THEN OUTPUT
   "TRUE OUTPUT "FALSE
END
```

4. Hallar una superficie utilizando el método Monte Carlo:

```
TO MC
   DRAW PU MAKE "IN 0
   MC1 1000 10 100
   (PRINT [LA SUPERFICIE ES] (:IN))
END

TO MC1 :N :XN :YN
   IF :N=0 THEN STOP
   PUNTO.ALEATORIO :XN :YN
   IF DENTRO? THEN MAKE "IN :IN+1
   MC1 :N – 1 :XN :YN
END

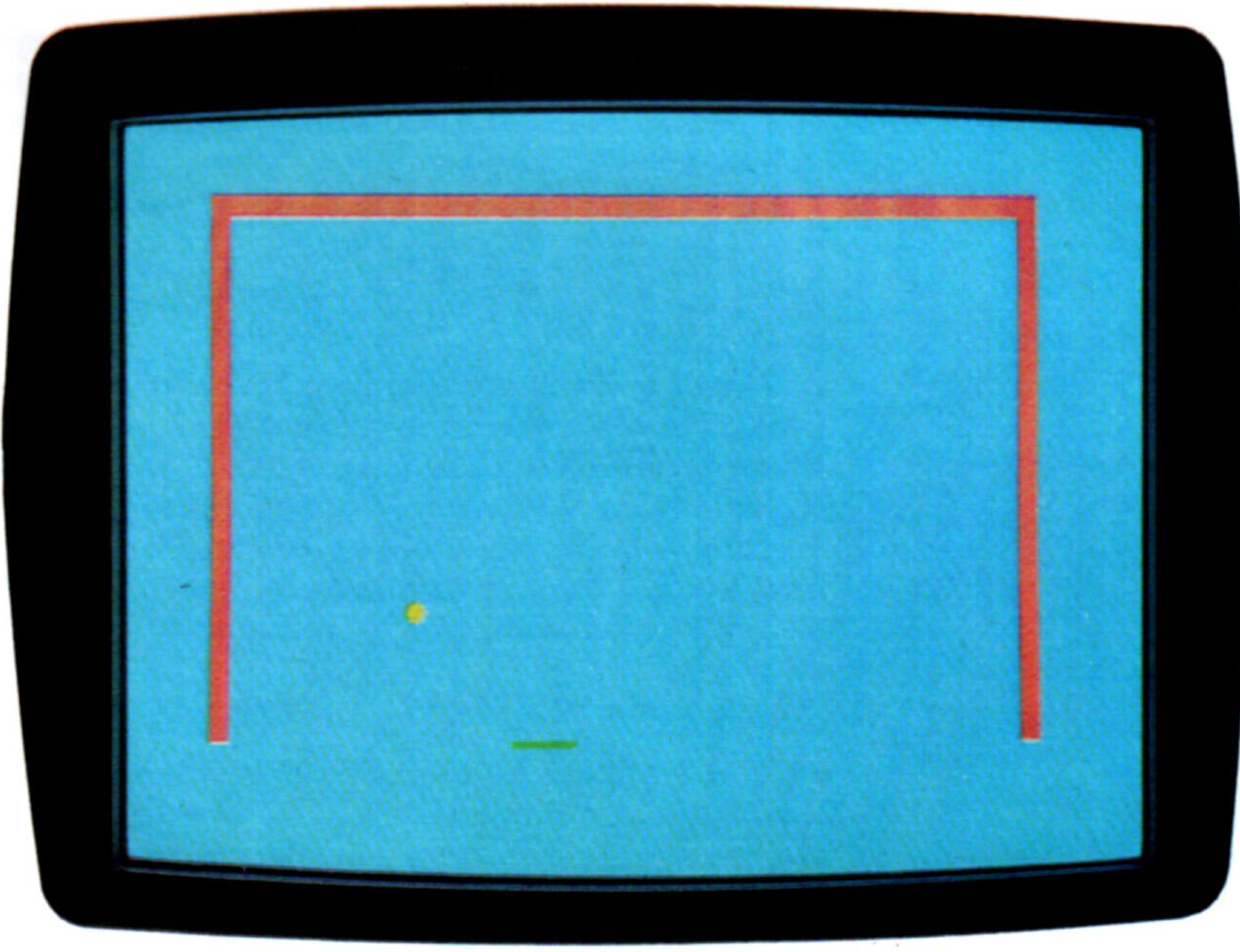
TO PUNTO.ALEATORIO :XN :YN
   SETXY RANDOM :XN RANDOM :YN
END

TO DENTRO?
   IF YCOR< XCOR*XCOR THEN OUTPUT
   "TRUE OUTPUT "FALSE
END
```

Squash

¿Por qué no hacer un poco de deporte ante su pequeña pantalla? Esta versión de «Squash» ha sido escrita para el microordenador Thomson TO 7

Gracias a su ordenador, puede jugar a squash cómodamente sentado en un sillón. La raqueta se desplaza con la ayuda de la palanca de mando o de las teclas Q, S y la barra espaciadora. Dispone de diez pelotas que ha de mantener en juego el mayor tiempo posible. Cada pelota que se devuelve proporciona un punto.



```
10 REM **********
20 REM * SQUASH *
30 REM **********
40 CLEAR ,,2
50 GOSUB 660
60 GOTO 180
70 D=2* ((STICK (0)=7)-(STICK (0)=3))
80 IF D<>0 THEN DO=D
90 IF STICK (0)=0 THEN DO=0
100 RX=RX+DO
110 RETURN
120 D$=INKEY$
130 D=2* ((D$="Q")-(D$="S"))
140 IF D<>0 THEN DO=D
150 IF D$=N$ THEN DO=0
160 RX=RX+DO
170 RETURN
180 LOCATE BX,BY
190 COLOR 3,6
200 PRINT N$;
210 BX=BX+DX
220 BY=BY+DY
230 LOCATE BX,BY
240 PRINT B$;
250 IF BY=22 AND ABS (BX-RX-3) >1 THEN 360
260 IF BY=22 THEN S=S+1:BEEP:DY=-DY
270 IF BY=1 THEN BEEP:DY=-DY
280 IF BX=2 OR BX=37 THEN BEEP:DX=-DX
290 ON JS GOSUB 70,120
300 IF RX<0 THEN RX=0
310 IF RX>33 THEN RX=33
320 LOCATE RX,RY
330 COLOR 2
340 PRINT R$;
350 GOTO 180
360 NB=NB+1
370 IF NB=11 THEN 480
380 LOCATE BX, BY
390 PRINT N$;
400 FOR I=1 TO 3
410 BEEP
420 FOR J=1 TO 100
430 NEXT J
440 NEXT I
450 DO=0
460 GOSUB 940
470 GOTO 180
480 LOCATE 13,5
490 COLOR 0
500 PRINT "PUNTUACION :";S;
510 IF S>R1 THEN R1=S
520 LOCATE 13,10
530 PRINT "RECORD :";R1;
540 LOCATE 13,15
550 PRINT "OTRA ?";
560 NB=0
570 S=0
580 D$=INKEY$
590 IF D$<>" " THEN 580
600 D$=INKEY$
610 IF D$=" " THEN 600
620 IF D$<>"N" THEN 50
630 SCREEN 4,6,6
640 CLS
650 END
660 CLS
670 SCREEN 1,6,6
680 ATTRB 1,1
690 DEFINT A-Z
700 DEFGR$ (0)=255,255,255,0,0,0,0,0
710 DEFGR$ (1)=24,126,126,255,255,126,126,24
720 N$=CHR$(32)
730 LOCATE 1,10,0
740 PRINT "JOYSTICK (S o N) ?";
750 D$=INKEY$
760 IF D$=" " THEN 750
770 IF D$="S" THEN JS=1 ELSE JS=2
780 CLS
790 ATTRB 0,0
800 COLOR 6,1
810 FOR BX=1 TO 38
820 LOCATE BX,0
830 PRINT N$;
840 NEXT BX
850 FOR BY=1 TO 22
860 LOCATE 1,BY
870 PRINT N$;
880 LOCATE 38,BY
890 PRINT N$;
900 NEXT BY
910 R$=N$+N$+GR$ (0)-GR$ (0)+GR$ (0)+N$+N$
920 B$=GR$ (1)
930 RX=16
940 RY=23
950 BY=22
960 BX=INT (RND*34)+3
970 DY=-1
980 DX=(INT (RND*2)-0.5)*2
990 RETURN
```

A prueba de error

Vamos a poner un ejemplo de diseño de arriba abajo («top-down»): un programa en assembly para depurar y poner a punto otros programas («debugger»)

Veamos primeramente las fases de especificación y diseño de que se compone la obtención de un eliminador de errores. La especificación es bastante inmediata; ya tuvimos ocasión de considerar las funciones que suponemos se incorporarán en el programa (véase p. 1219).

Inputs del programa:

1. *Un programa a depurar:* Supondremos que ya está cargado en la memoria, junto con el depurador.
2. *Órdenes:* Debemos decidir si las órdenes se introducirán directamente o por medio de opciones de menú. Entraremos órdenes de un solo carácter según la tabla al margen.
3. *Direcciones:* Como pueden ser introducidas en hexadecimal, habrá que convertir una cadena de dígitos hexadecimales ASCII en números binarios de 16 bits.

Outputs del programa:

1. *"Ecos" de los caracteres entrados:* Téngase en cuenta que por el hecho de apretar una tecla no se obtiene un carácter en la pantalla. El ordenador debe ser programado para que haga eso.
2. *Números de 8 y 16 bits:* Son aceptados en forma de cadenas de números hexadecimales.
3. *Cadenas:* Que sirvan como etiquetas de los números anteriores.

B	insertar Punto de Ruptura (*B*reakpoint)
U	eliminar Punto de Ruptura (*U*n-insert)
D	visualizar (*D*isplay) Puntos de Ruptura en curso
S	comenzar (*S*tart) ejecución programa
G	ir (*G*o). Reemprender ejecución programa
R	visualizar contenido *R*egistros
M	inspeccionar y cambiar posición de *M*emoria
Q	salir (*Q*uit)

Hay muchas formas de dividir un programa en módulos y éstos en subrutinas, pero siempre debe proporcionarse un módulo "externo" que sirva de caparazón (*shell*) o enlace de los restantes. En nuestro programa este módulo tomará la siguiente forma:

Módulo principal

Data:

- **Dirección de inicio** del programa (16 bits)
- **Aviso** para la entrada de órdenes (el carácter ASCII individual '>')
- **Carácter de Orden**, está en ASCII y es individual (¿usaremos caracteres en minúscula?)
- **Dirección de Ruptura**, es la dirección de la rutina administradora de la interrupción SWI

Proceso:

```
Establecer la Interrupción
GET (tomar) la Dirección de Inicio
REPEAT
   DISPLAY aviso
REPEAT
   Tomar la Orden
UNTIL que la Orden sea válida
DISPLAY la Orden (eco)
IF Orden = 'B' THEN
   Insertar Punto de Ruptura
ELSE IF Orden = 'U' THEN
   Eliminar Punto de Ruptura
ELSE IF...
   Hasta que Orden = 'Q'
```

Fin del módulo principal

Con esto ya se tiene una buena idea de las rutinas que harán falta. No es lo mismo un módulo que una rutina. Hay varias rutinas que pueden agruparse lógicamente por compartir los mismos datos: por ejemplo, un módulo en nuestro caso tratará los puntos de ruptura. La fase siguiente muestra cómo es posible diseñar este módulo:

Módulo de rupturas

Data:

- **Tabla-Puntos-Ruptura** es una matriz de direcciones de 16 bits que almacenará las direcciones de los puntos de ruptura
- **Valores-Sustituidos** es una matriz de valores de 8 bits relativos a la tabla anterior. Aquí pueden almacenarse los opcodes que son sustituidos por una instrucción SWI en el punto de ruptura
- **Número-Puntos-Ruptura** es un valor de 8 bits que da el número de puntos de ruptura activos
- **Punto-Ruptura-Siguiente** es un valor de 8 bits, que contiene el siguiente punto de ruptura según avanza la ejecución
- **SWI-Opcode** es un opcode de 8 bits para la instrucción SWI

Proceso 1: Inserción Puntos-Ruptura

```
IF Número-Puntos-Ruptura < MAX THEN
   Tomar-Dirección
   Añadir 1 a Número-Puntos-Ruptura
   Almacenar Dirección en Tabla-Puntos-Ruptura
   (Número-Puntos-Ruptura)
ENDIF
```

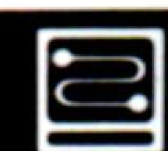
Fin del proceso 1

Proceso 2: Establecer-Punto-Ruptura (N)

```
(N nos dirá cuál de los puntos de la tabla ha de ser
activado)
   Tomar-Dirección en Tabla-Puntos-Ruptura (N)
   Tomar el Opcode contenido en esa Dirección
   Almacenarlo en Valores-Sustituidos (N)
   Almacenar el Opcode SWI en Dirección
```

Fin del Proceso 2

El Proceso 2 se encuentra en una fase en la que podríamos empezar a codificar. Cuatro son los valores de datos que han de ser manipulados: N, el parámetro indicador del punto de ruptura que hay que usar, es un número de 8 bits que emplearemos como desplazamiento en las dos tablas y que oscilará entre uno y Número-Puntos-Ruptura−1. Nóte-



se, sin embargo, que una tabla toma valores de 16 bits mientras la otra los toma de 8 bits. Asumiremos que N es pasado en A. La dirección del punto de ruptura tomada de dicha tabla se introducirá en X. El opcode sustituido será llevado a B para su introducción en la tabla Valores-Sustituidos. Así B puede ser empleado para colocar el opcode SWI en la dirección apropiada.

Proporcionamos aquí la codificación final del Proceso 2 (Módulo Establecer-Punto-Ruptura); la siguiente tarea consistirá en diseñar un módulo para manejar las entradas y salidas. De lo que llevamos dicho, se puede colegir que existen diferentes tareas de E/S a ejecutar por el programa. Por el momento, asumiremos la existencia de dos subrutinas: INCH, encargada de introducir un carácter individual en el registro A a partir del teclado; y OUTCH, que enviará el carácter desde A hasta la posición de la pantalla que indique el cursor. Este módulo requiere las siguientes subrutinas:

1. TomarOrden: Entra desde el teclado la orden siguiente.
2. TomarDirección: Toma del teclado una dirección hexadecimal de 1 a 4 caracteres de longitud.
3. TomarValor: Toma un valor hexadecimal de 1 o 2 caracteres de longitud para modificar el valor de una posición de memoria.
4. VisualizarValor: Visualiza en pantalla un valor hexadecimal de dos caracteres.
5. VisualizarDirección: Visualiza en pantalla una dirección hexadecimal de cuatro caracteres.

Nuestro enfoque muestra la diferencia entre los métodos de programación de arriba abajo (*top-down*) y de abajo arriba (*bottom-up*). El primero puede llevarnos a definir y codificar estas operaciones de manera independiente, acabando por escribir rutinas separadas que esencialmente hacen la misma cosa.

El método ascendente puede ahorrarnos tiempo, espacio y esfuerzo, al escribir sencillamente unas cuantas rutinas útiles que emplearemos en distintas circunstancias.

Estas rutinas son:

GETCH: Entrar un carácter individual en A, comprobándolo en una lista de caracteres válidos (letras de órdenes o dígitos hexa), haciendo el eco de los caracteres válidos e ignorando los restantes.
GETHX2: Sirve para que GETCH tome dos dígitos hexa y los convierta en un número de ocho bits.
GETHX4: Toma cuatro dígitos hexa y forma un número de 16 bits.
PUTHEX: Visualiza un número de 8 bits como dos dígitos hexa (puede ser llamada dos veces para que visualice un número de 16 bits).
PUTCR: Para sacar un carácter de retorno de carro (o retorno de carro y salto de línea si es necesario).

Cada una de estas rutinas han de ser desarrolladas. Consideremos primero el diseño de GETCH.

La rutina GETCH

Data:

```
   Car-In es un carácter ASCII entrado desde teclado
      (contenido en A)
   Car-Válidos retiene la dirección de 16 bits de la tabla
      de caracteres válidos
   Número-Car-Val es un valor de 8 bits
   Car-Inspec es un contador de 8 bits
Proceso:
   REPEAT
      Tomar el siguiente Car-In
      Poner Car-Inspec a (Número-Car-Val-1)
      While Car-Válidos (Car-Inspec) <> Car-In
         AND Car-Inspec > =0
         Decrementar Car-Inspec
   Until Car-Inspec > = 0
   DISPLAY Car-In
```

Para codificar esto, debemos emplear A para almacenar Car-In, y el valor de 16 bits de Car-Válidos puede ser pasado y guardado en X. En B puede ser pasado Número-Car-Val, pero necesita una mayor permanencia, lo que conseguiremos llevándolo a la pila. En este caso podemos emplear B para Car-Inspec. Observe que B devolverá el desplazamiento a la tabla, lo que será útil para interpretar las órdenes y para la conversión hexadecimal.

He aquí la forma final de esta rutina codificada. En la siguiente lección desarrollaremos las restantes rutinas necesarias para el módulo de entrada-salida.

Rutina GETCH

```
GETCH    PSHS   B         Salva B
REPTOO   BSR    INCH      Toma el siguiente Car-In
         LDB    1,S       Activa Car-Inspec
         DECB             Resta 1 del desplaz. máximo
WHILOO   BLT    ENDWOO    Mientras Car-Inspec>=0
         CMPA   B,X       AND
         BEQ    ENDWOO    Car-In<>Car-Válidos(Car-
                          Inspec)
         DECB             Decrementa Car-Inspec
         BRA    WHILOO
ENDWOO   TSTB
UNTLOO   BLT    REPTOO    Hasta que Car-Inspec>=0
         BSR    OUTCH     Visualiza Car-In
         LEAS   1,S       Incrementa S para "olvidar"
                          el valor primitivo de B
         RTS
```

Módulo Estab.-Punto-Ruptura

```
                               Declaraciones de datos
TABPR    RMB    32                Tabla-Puntos-Ruptura
TABVS    RMB    16                Valores-Sustituidos
NUMPR    FCB    0                 Número-Puntos-Ruptura
NEXTPR   FCB    0                 Siguiente-Punto-Ruptura
OPSWI    FCB    $3F               Opcode-SWI
MAXPR    FCB    16                Núm. máx. de Puntos-Ruptura
                               Proceso 2—Estab.-Punto-Ruptura
PRO2     PSHS   B,X               Guarda regs. que se alterarán
         LSLA                     Multiplica por dos el desplaz.
         LEAX   TABPR,PCR         Dirección de base de la tabla
         LDX    A,X               Toma Dir. en Tabla-Puntos-Rup. (N)
         LDB    ,X                Toma en esa dir. el Opcode
         LSRA                     Restaura valor primitivo A
         STB    A,X               Almacena Opcode en Valores-
                                  Sustituidos (N)
         LDB    OPSWI,PCR         Toma el Opcode SWI
         STB    ,X                Lo almacena en la dir.
         PULS   B,X,PC            Restaura y retorna
                               Fin del proceso 2
```

Alerta roja

«Flyerfox» es un vertiginoso juego que permite al usuario participar en combates aéreos desde la cabina de un moderno avión de combate

A pesar de que los juegos por ordenador han recorrido un largo camino desde los primeros días de los *Space invaders* (véase p. 1095), la mayor parte de sus innovaciones han estado relacionadas con el desarrollo de gráficos. Los programadores se han preocupado fundamentalmente de hallar nuevas formas de comprimir los datos de un número cada vez mayor de pantallas de gráficos en una cantidad de RAM limitada. Mientras tanto, las excelentes capacidades de sonido de muchos ordenadores personales han sido en gran parte dejadas de lado.

Ahora la empresa norteamericana Tymac ha empezado a distribuir una serie de juegos que incorporan síntesis de voz sin utilizar en realidad ninguna interface especial. El primero de estos programas que se ha comercializado a nivel masivo es *Flyerfox*, para el Commodore 64. Se trata de un programa de simulación de vuelo en el cual el jugador "pilota" un avión de combate que debe escoltar a un Jumbo, a bordo del cual viaja un alto funcionario del gobierno, a través de un espacio aéreo en litigio. Los cazas enemigos intentan derribar el avión comercial, y el objetivo del jugador es entrar en combate con éstos y abatirlos. La síntesis de voz utilizada en el juego consiste en una serie de mensajes que se le transmiten al jugador-piloto desde el Jumbo.

El sintetizador de voz es una parte del software que ocupa alrededor de 11 K de memoria para almacenar los datos que se utilizan para reproducir las frases requeridas. *Flyerfox* emplea el método de codificación de "predicción lineal". En este sistema, las palabras se convierten en señales digitales, que se almacenan luego en RAM. Cuando se necesita una palabra determinada, se accede a los datos digitales correspondientes y la palabra se reproduce a través del chip SID del Commodore.

Los gráficos del juego son siempre en alta resolución. La visualización en pantalla consiste en vistas frontales que muestran el cielo tal como se vería desde la ventana de la cabina, y el panel de instrumentos. El jugador cuenta con varios medios auxiliares de navegación, entre ellos una brújula y un panel de radar que muestra a los cazas MiG aproximándose, al tiempo que se le concede tiempo para prepararse para el combate. Otra de las ayudas que se proporcionan son dos luces intermitentes, una a cada lado del horizonte, que le indican si los MiG se hallan sobre o bajo el nivel de la cabina.

Las escenas de combate son rápidas y muy realistas. Cuando aparece un MiG en la pantalla, el programa produce un zumbido de aviso y el jugador debe entonces maniobrar el *Flyerfox* de modo tal que el atacante quede justo en el punto de mira. Esto no es fácil, ya que los aviones hacen regates y bajan en picado a gran velocidad. Una vez el objetivo está fijado en la mira, el jugador puede disparar los misiles rastreadores de calor; sin embargo, éstos no son infalibles y con frecuencia los cazas enemigos consiguen escapar.

A pesar de que los gráficos son de una gran calidad, no ofrecen excesiva variedad. La ilusión de movimiento se consigue modificando los patrones de las nubes, y el suelo es simplemente una cuadrícula rotatoria. También hay que decir que el avión de línea aporta poco o nada al juego. Dejando de lado el evidente paralelismo con el hecho real del Jumbo 007 surcoreano abatido por un avión soviético hace un tiempo, que deja el juego abierto a ciertas acusaciones de falta de buen gusto, resulta difícil comprender los motivos de la inclusión del avión comercial. Sólo se lo puede observar desde la cola, y el *Flyerfox* es incapaz de darle alcance cuando intenta entrar en combate con los cazas. Además, a diferencia de un avión verdadero, al ser atacado, el Jumbo ni siquiera intenta huir.

Ciertamente, *Flyerfox* representa una nueva tendencia en materia de juegos por ordenador. La síntesis de voz dentro de un programa es un tema que se ha estado considerando durante cierto tiempo. Ahora Tymac ha producido un juego que utiliza la voz, pero que no exige ninguna interface ni dispositivo de hardware especial. En este sentido, es probable que llegue a ser considerado como un hito en el desarrollo de juegos por ordenador.

Perseguir y destruir
El piloto del Flyerfox puede reunir información acerca de las posiciones de los cazas enemigos tanto a partir del panel de instrumentos como de la observación del cielo. Los puntos de la pantalla del radar son aviones, si bien no todos ellos atacarán. La altura relativa de los MiG se indica mediante los dos cuadrados blancos que se observan a cada uno de los lados del altímetro. La brújula sirve para volver a localizar el Jumbo

Los MiG atacan

Ian McKinnell

Flyerfox: Para el Commodore 64
Editado por: Tymac Corporation
Autores: Gregory Carbonaro, Charles Teufert, Ronald Pintus, Arthur Aspromatis
Palanca de mando: Necesaria
Formato: Disco o cassette

Editorial Delta, S.A.



10062
9 788485 822836